Variational Quantum Algorithms

Full Course

hbar.university

Module 1 — Hybrid Quantum Classical Systems

The Variational Principle

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.1

This is **the theorem that makes VQAs legal**. Everything downstream in this course — every cost function, every optimizer, every claim about "approximating the ground state" — depends on this one result. It is the justification for treating the scalar number $\langle\psi|H|\psi\rangle$ as something worth minimizing.

Formal Statement

Let H be a Hermitian operator on a Hilbert space $\mathcal{H}$, with eigenvalues $E_0 \leq E_1 \leq E_2 \leq \ldots$ and corresponding eigenstates $|\phi_0\rangle, |\phi_1\rangle, \ldots$. For any normalized state $|\psi\rangle \in \mathcal{H}$,

$$\langle\psi|H|\psi\rangle \geq E_0,$$

with **equality if and only if** $|\psi\rangle = |\phi_0\rangle$ (up to a global phase, and modulo degeneracy).

This is the **Rayleigh-Ritz variational principle**, stated in its quantum-mechanical form. It is the oldest member of a family of inequalities that appear across mathematical physics whenever a bilinear form has to be bounded below by its smallest eigenvalue.

Proof in Three Lines

Expand $|\psi\rangle = \sum_i c_i|\phi_i\rangle$ in the eigenbasis of H, with $\sum_i |c_i|^2 = 1$. Then

$$\langle\psi|H|\psi\rangle = \sum_i |c_i|^2 E_i \geq E_0 \sum_i |c_i|^2 = E_0,$$

because every $E_i \geq E_0$ and $\sum_i |c_i|^2 = 1$. Equality requires all weight in the lowest eigenvalue component.

That is the entire proof. Every word of Module 5's "not bias" polemic, every gradient descent step in a VQE run, every barren plateau theorem — all of it lives above this three-line argument.

Intuition

The variational principle says: **you cannot accidentally overshoot the ground state.**

Any guess $|\psi\rangle$ you make about the ground state gives you an **upper bound** on the true ground-state energy. A better guess gives a tighter bound. The best guess reachable by your ansatz gives you the tightest bound your ansatz can produce.

This is the feature that makes VQAs optimizable at all. Without the variational principle, minimizing $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ would just be minimizing some scalar function with no guarantee that doing so gets you closer to anything physical. With it, every decrease in $C(\boldsymbol{\theta})$ is a decrease in your upper bound on the real thing.

Another way to say the same thing: **energy estimates from variational methods are biased low in a useful direction.** They are always at least E_0, so they can never lie to you in the "your answer is lower than the ground state" direction. The only way to be wrong is to be too high — which just means your ansatz isn't expressive enough to reach the true ground state. That is a diagnosable failure mode, not a silent one.

Structural Role

This node is the **foundation stone** of Module 1. Everything else in the module specializes, implements, or complicates the picture laid out here:

- Node 1.2 (architecture) tells you how the variational principle gets executed in practice on a hybrid device.
- Node 1.3 (expectation estimation) tells you how you actually **measure** $\langle\psi|H|\psi\rangle$ when ψ lives on a real quantum computer.
- Node 1.8 (quantum subsystem as objective generator) reframes the variational principle in thesis language: the quantum device implements a stochastic oracle for a quantity the variational principle tells you is worth bounding.

If you forget this node, nothing else in the course makes sense. If you remember only this node, you still know why VQAs exist.

Relations to Other Concepts

- **Rayleigh quotient** — the classical version from linear algebra: for a symmetric matrix A, $\min_{x\neq 0} \frac{x^T A x}{x^T x} = \lambda_{\min}(A)$. The quantum variational principle is this statement on unit vectors in $\mathcal{H}$.
- **Hartree-Fock method** — the first and most famous non-quantum-computer application of the variational principle to chemistry.
- **Density Matrix Renormalization Group (DMRG)** — also a variational method, restricted to matrix-product-state ansätze.
- **Reinforcement learning policy gradient** — structurally analogous: minimize an upper bound on an expected return by descending its stochastic gradient.

Worked Example — The Simplest Variational Calculation

Let $H = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ (Pauli X). Its eigenvalues are ± 1, so $E_0 = -1$.

Consider the one-parameter ansatz

$$|\psi(\theta)\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle.$$

Compute

$$C(\theta) = \langle\psi(\theta)|X|\psi(\theta)\rangle = 2\cos(\theta/2)\sin(\theta/2) = \sin\theta.$$

The minimum over θ is -1, achieved at $\theta = -\pi/2$. This equals E_0 exactly — the ansatz is expressive enough to reach the true ground state. The variational principle guarantees $C(\theta) \geq -1$ for all θ, and direct calculation confirms it.

Now try a less expressive ansatz: $|\psi(\theta)\rangle = \cos(\theta)|0\rangle + \sin(\theta)|+\rangle$ (normalized properly — exercise). Compute $C(\theta)$ and show that its minimum is strictly greater than -1 — the ansatz cannot reach the true ground state, and the variational principle still holds.

Visualization

Pending. Diagram: a horizontal line at E_0. Above it, a scatter of points representing $C(\boldsymbol{\theta})$ values for random $\boldsymbol{\theta}$; all points strictly above the line. An arrow showing the optimizer "pushing down" toward E_0 but never crossing it.

Exercises

1. Prove that if H has a degenerate ground space, then $C(\boldsymbol{\theta}) = E_0$ implies $|\psi(\boldsymbol{\theta})\rangle$ lies in the ground space, but not necessarily that it equals any particular ground-state vector.
2. Show that $\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$ is always achieved (i.e., the infimum is attained) when the parameter domain is compact and C is continuous. Why are these assumptions both needed?
3. (VQA) Give an example of a two-qubit Hamiltonian and a one-parameter ansatz for which $\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) > E_0$ strictly. What does this tell you about the expressivity of your ansatz?

Research Extensions

- **Variational bounds for excited states** — the SSVQE and VQD algorithms extend the principle to find $E_1, E_2, \ldots$ by enforcing orthogonality constraints.
- **Krylov subspace methods** — classical methods that build better variational subspaces iteratively from matrix-vector products.
- **Witness operators** — in entanglement theory, operators whose expectation values bound quantities of interest in the same structural sense.

Cross-links to Other Courses

- **Linear Algebra** — Rayleigh quotients, min-max characterization of eigenvalues.
- **Quantum Foundations** — the variational principle as a corollary of the spectral theorem.
- **Statistics Module 10** — structurally analogous upper-bound arguments in model selection (e.g., PAC bounds as upper bounds on true risk).

The Quantum-Classical Architecture

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.2

This node describes the **operational shape** of a variational quantum algorithm: two coupled layers, one quantum and one classical, exchanging information through a tight feedback loop. Everything in Module 1 from here on lives inside one or the other of these two layers.

Node 1.1 justified *why* we care about the number $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$. This node describes *the machine that computes it and decides where to go next.*

The Two Layers

A variational quantum algorithm is built from two distinct subsystems:

The quantum layer. Takes a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^P$ as input. Prepares a parameterized state $|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})|0\rangle^{\otimes n}$ on a real or simulated quantum device. Measures a Hamiltonian H to return a noisy scalar estimate $\widehat{C}(\boldsymbol{\theta}) \approx \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$. That is its entire job.

The classical layer. Takes the current parameter vector $\boldsymbol{\theta}_k$ and the scalar feedback $\widehat{C}(\boldsymbol{\theta}_k)$ as input. Produces a new parameter vector $\boldsymbol{\theta}_{k+1}$ according to some update rule:

$$\boldsymbol{\theta}_{k+1} = \mathcal{A}\big(\boldsymbol{\theta}_k, \widehat{C}(\boldsymbol{\theta}_k), \mathcal{H}_k\big),$$

where $\mathcal{A}$ is the optimizer and $\mathcal{H}_k$ is its internal memory (momentum, particle populations, trust-region radii, whatever). That is **its** entire job.

Neither layer understands the other directly. The quantum layer does not know it is being optimized; it is a stateless oracle. The classical layer does not know what is happening inside the quantum device; it sees only numbers.

The Feedback Loop

Execution alternates between the two layers. At iteration k:

1. The classical layer hands $\boldsymbol{\theta}_k$ to the quantum layer.
2. The quantum layer prepares $|\psi(\boldsymbol{\theta}_k)\rangle$, measures, and returns $\widehat{C}(\boldsymbol{\theta}_k)$.
3. The classical layer consumes the feedback, updates its internal state, and produces $\boldsymbol{\theta}_{k+1}$.
4. Repeat.

This is a **closed-loop discrete-time control system**. The "plant" is the quantum objective oracle. The "controller" is the classical optimizer. The "reference signal" is implicit: reduce $\widehat{C}$.

Framing VQA this way is not just metaphor. It means you can import every concept from classical control theory — stability, observability, disturbance rejection — and apply it to VQA directly. Module 7 develops this in detail.

Why Hybrid and Not All-One-Or-The-Other

Why not fully quantum? A fully quantum algorithm for ground-state estimation (quantum phase estimation, for example) requires circuit depth and coherence time that NISQ devices do not have. Fault-tolerant quantum computing is the long game. VQAs are the near-term workaround: use a shallow quantum circuit to generate a stochastic cost, and spend the classical compute budget on optimization.

Why not fully classical? A fully classical simulation of $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ requires storing the state vector, which is exponential in qubit count. For 50+ qubits this is infeasible. The whole point of running the circuit on quantum hardware is that state preparation is efficient on a quantum device regardless of qubit count — the exponential cost shows up in **measurement**, not in **storage**.

The hybrid design is the minimum-viable use of quantum hardware given those two constraints. You run the part of the computation that classical computers cannot do (preparing a superposition of 2^n amplitudes) on the quantum device, and you run everything else — parameter selection, state tracking, decision logic — on a classical computer that is actually good at that.

Structural Role

This node is the **scaffold** onto which the rest of Module 1 bolts.

- Nodes 1.3, 1.4 zoom into the quantum layer and explain what "measuring" and "estimating a gradient" actually mean in practice.
- Node 1.5 zooms into the classical layer and describes what the optimization landscape looks like from the inside.
- Node 1.6 reframes the whole architecture as a system-identification problem.
- Nodes 1.7, 1.8, 1.9 re-describe the same architecture in the thesis vocabulary (epistemic vs ontological, objective generator, separation of concerns).

All of those nodes assume you understand the two-layer picture introduced here.

Relations to Other Concepts

- **Model-based vs model-free reinforcement learning** — VQA is model-free: the optimizer never builds a model of the quantum landscape, it just queries it.
- **Bayesian optimization** — a different family of hybrid systems where the classical side **does** build an explicit surrogate of the black-box function.
- **Simulation-based inference** — classical ML with a simulator as the oracle; structurally similar to VQA with the quantum device replaced by a (typically deterministic) simulator.
- **Control-plant decomposition** — the standard block-diagram of classical control theory maps onto VQA cleanly (see Module 7).

Worked Example — A Single Iteration

Suppose you have a 4-qubit hardware-efficient ansatz with 8 parameters. At iteration $k = 0$:

1. You initialize $\boldsymbol{\theta}_0$ randomly in $[-\pi, \pi]^8$.
2. You send $\boldsymbol{\theta}_0$ to the quantum layer.
3. The quantum layer compiles the circuit, runs 1024 shots, measures in the eigenbases of the Pauli decomposition of H, and returns $\widehat{C}(\boldsymbol{\theta}_0) \approx -0.42$.
4. The classical layer is running ADAM. It needs a gradient, so it queries the quantum layer 16 more times (parameter-shift rule, 2 evaluations per parameter, 8 parameters) to get $\widehat{\nabla} C(\boldsymbol{\theta}_0)$.
5. ADAM produces $\boldsymbol{\theta}_1$.

Iteration 0 has cost: 17 circuit evaluations, $17 \times 1024 = 17{,}408$ shots. That is the **per-iteration quantum resource budget**, and it is the real currency of VQA. (See node 1.4 for why the parameter-shift rule gives exact-up-to-noise gradients, and Module 2 for how to design circuits that make this cheap.)

Visualization

Pending. Block diagram: two boxes labeled "Classical Optimization Layer" (top) and "Quantum Objective-Generating Substrate" (bottom). Arrows: down-arrow labeled $\boldsymbol{\theta}_k$, up-arrow labeled $\widehat{C}(\boldsymbol{\theta}_k)$. Dashed box around both labeled "VQA".

Exercises

1. Given a 10-qubit ansatz with 30 parameters and a Hamiltonian with 50 Pauli terms, estimate the number of shots needed per iteration for a gradient-based optimizer using the parameter-shift rule. State your assumptions about shots-per-expectation.
2. Suppose the classical optimizer has unbounded compute but the quantum layer is rate-limited to 1000 circuit evaluations per hour. How does this constrain the choice of optimizer?
3. Why is it incorrect to say "the quantum device is just a subroutine that evaluates $C(\boldsymbol{\theta})$"? What does this framing hide about the cost and noise structure of VQA?

Research Extensions

- **Measurement-adaptive VQA** — closing a second feedback loop inside the quantum layer, where the choice of measurement basis depends on previous outcomes (see Module 10).
- **Latency-aware optimizer design** — choosing optimizers based on the classical-quantum communication overhead rather than iteration count.
- **Co-processing architectures** — tighter hardware integration between quantum and classical components (e.g., IBM's runtime primitives, Quantinuum's TKET runtime).

Cross-links to Other Courses

- **Control Theory** — the feedback-loop framing; see Module 7 for the full mapping.
- **Internet Systems** — request-response latency considerations for cloud-accessed quantum hardware.
- **Systems Thinking** — composition of subsystems with different time scales and observability.

Expectation Value Estimation

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.3

This node answers the question that 1.1 raised and 1.2 deferred: **how do you actually get the number $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ out of a quantum computer?** The answer is "you don't — you get a noisy estimate of it, and the noise is intrinsic to how measurement works."

This is where "shot noise" enters the story. Everything about VQA optimization that distinguishes it from deterministic classical ML ultimately traces back to this node.

The Three-Step Recipe

Step 1: Decompose. Write the Hamiltonian as a weighted sum of Pauli strings:

$$H = \sum_{j=1}^{M} h_j P_j,$$

where each $P_j \in \{I, X, Y, Z\}^{\otimes n}$ is a tensor product of single-qubit Pauli operators and $h_j \in \mathbb{R}$. This decomposition exists for any Hermitian H — the Pauli strings form a basis for the space of Hermitian operators on n qubits.

Step 2: Measure each term. By linearity of expectation,

$$\langle\psi|H|\psi\rangle = \sum_{j=1}^{M} h_j \langle\psi|P_j|\psi\rangle.$$

So you can measure $\langle P_j\rangle$ separately for each j and sum the results. Each measurement requires rotating into the eigenbasis of P_j and then measuring in the computational basis. For a Pauli string like $X_1 Z_2$, you apply a Hadamard on qubit 1, do nothing on qubit 2, and measure both.

Step 3: Sample. Measurement in quantum mechanics is a random process. For N shots, you get N independent samples $s_j^{(1)}, \ldots, s_j^{(N)} \in \{-1, +1\}$ distributed according to the eigenvalue probabilities of P_j on $|\psi\rangle$. The unbiased estimator is the sample mean:

$$\widehat{\langle P_j\rangle} = \frac{1}{N} \sum_{i=1}^{N} s_j^{(i)}.$$

Combine across terms to get

$$\widehat{C}(\boldsymbol{\theta}) = \sum_{j=1}^{M} h_j \widehat{\langle P_j\rangle}.$$

That is the number the quantum layer returns to the classical optimizer.

Why This Is Noisy

Every step introduces uncertainty that does not exist in a classical function evaluation.

Shot noise. Even on a perfect device, $\widehat{\langle P_j \rangle}$ is a sample mean and therefore has variance. For a Pauli observable with true expectation value $p \in [-1, 1]$,

$$\mathrm{Var}\big[\widehat{\langle P_j \rangle}\big] = \frac{1 - p^2}{N} \leq \frac{1}{N}.$$

Combined across terms, the total variance of $\widehat{C}(\boldsymbol{\theta})$ scales as $O(\|h\|_1^2/N)$ in the worst case. Halving the uncertainty costs four times as many shots.

Hardware noise. On real devices, additional sources kick in: gate errors, decoherence during state preparation, readout errors, calibration drift. These are not sampling noise — they corrupt the prepared state before measurement — but from the optimizer's perspective they look similar: the number coming back is not the ideal $C(\boldsymbol{\theta})$.

Basis-change overhead. Measuring M Pauli terms naively requires M separate circuits (one per measurement basis). For molecular Hamiltonians, M can be in the thousands. This motivates **measurement grouping** — partitioning Pauli strings into commuting sets that can be measured simultaneously (Module 2).

What the Optimizer Actually Sees

From the classical layer's perspective, each call to the quantum layer returns a random variable:

$$\widehat{C}(\boldsymbol{\theta}) = C(\boldsymbol{\theta}) + \varepsilon(\boldsymbol{\theta}),$$

where $C(\boldsymbol{\theta})$ is the ideal expectation value and $\varepsilon(\boldsymbol{\theta})$ is a zero-mean noise term with variance $\sigma^2(\boldsymbol{\theta})$ that depends on the number of shots, the structure of H, and (on real hardware) the physical noise model.

This is a **stochastic oracle** in the sense of classical optimization theory. The optimizer never sees $C(\boldsymbol{\theta})$ directly, only samples from its distribution. Every classical optimization algorithm used in VQA must be robust to this — or at least behave reasonably given noisy inputs.

Module 5 is essentially about how to make optimizers behave reasonably given ε. Module 9 is essentially about how to reduce ε at the source.

Structural Role

This is the node that makes everything else in VQA feel different from classical ML. Without shot noise, VQA would be straightforward stochastic optimization of a deterministic black-box function. With shot noise, you are doing stochastic optimization of a **stochastic** function whose sampling cost you also have to budget.

The shot budget is the real currency of NISQ-era VQA. Iterations are free; shots are not. Module 6 picks this up when discussing how to compare optimizers fairly (shots, not iterations, is the x-axis).

Relations to Other Concepts

- **Monte Carlo estimation** — the sample-mean estimator is a textbook Monte Carlo, with convergence rate $1/\sqrt{N}$.
- **Importance sampling** — can reduce variance by reshuffling the measurement distribution; used in advanced shot-allocation schemes.

- **Pauli grouping / qubit-wise commuting groups** — partitioning $\{P_j\}$ into simultaneously measurable sets.
- **Classical shadows** — a newer technique (Huang, Kueng, Preskill 2020) that estimates many observables from a single randomized measurement protocol, often dramatically cheaper than naive Pauli measurement.

Worked Example — Estimating $\langle Z \rangle$ on a Single Qubit

Suppose $|\psi\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle$. Then $\langle Z \rangle = \cos\theta$.

Run N shots. Each shot returns $+1$ with probability $\cos^2(\theta/2)$ and -1 with probability $\sin^2(\theta/2)$. The sample mean is an unbiased estimator of $\cos\theta$ with variance

$$\frac{1 - \cos^2\theta}{N} = \frac{\sin^2\theta}{N}.$$

Notice: the variance is **smallest at** $\theta = 0$ (where $\langle Z \rangle = +1$ deterministically) and **largest at** $\theta = \pi/2$ (where the state is $|+\rangle$ and measurement outcomes are maximally random). So the hardness of estimating a cost value depends on **where you are in parameter space**, not just how many shots you spend. This is one of the reasons variance-aware optimization (node 5.7) is a real thing.

Visualization

Pending. Diagram: top panel shows a single-qubit Bloch sphere with $|\psi\rangle$ at some angle θ. Middle panel shows 100 measurement outcomes as +1/-1 ticks. Bottom panel shows the running sample mean converging to $\cos\theta$ with a shrinking confidence band of width $\sim 1/\sqrt{N}$.

Exercises

1. Derive $\mathrm{Var}[\widehat{\langle P_j \rangle}] = (1 - p^2)/N$ from the definition of sample variance. What is the maximum variance over all possible $p \in [-1, 1]$?
2. Suppose $H = 0.5\,X + 0.5\,Z$. Compute $\mathrm{Var}[\widehat{C}(\boldsymbol{\theta})]$ as a function of $\boldsymbol{\theta}$ assuming independent shot budgets $N_X = N_Z = N$.
3. (VQA) If you have a total shot budget of 10,000 and two Pauli terms with coefficients $h_1 = 1.0$ and $h_2 = 0.1$, how should you allocate shots between them to minimize the variance of $\widehat{C}$? (Hint: Lagrange multipliers.)

Research Extensions

- **Classical shadows (Huang-Kueng-Preskill)** — exponentially cheaper estimation of many observables.
- **Adaptive shot allocation** — using the variance of early measurements to decide where to spend later shots.
- **Measurement grouping** — abelian Pauli partitioning; graph-coloring formulation.
- **Quantum-enhanced estimation** — amplitude estimation gives $1/N$ scaling instead of $1/\sqrt{N}$, but requires deeper circuits than NISQ can support.

Cross-links to Other Courses

- **Statistics Module 2** — sample mean, variance, standard error; this is literally that chapter applied to quantum measurements.
- **Quantum Foundations** — measurement postulate, projective measurements, Born rule.
- **Hardware Noise Models (Module 9)** — the other source of $\varepsilon(\boldsymbol{\theta})$.

Gradient Estimation

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.4

This node tackles the next question after "how do you evaluate $C(\boldsymbol{\theta})$": **how do you evaluate $\nabla C(\boldsymbol{\theta})$?**

The answer turns out to be remarkably clean for a wide class of circuits. It is called the **parameter-shift rule**, and it is one of the few genuinely beautiful results in VQA pedagogy. If you remember one technical result from Module 1 beyond the variational principle, remember this one.

The Problem

Gradient-based optimizers (SGD, Adam, L-BFGS, natural gradient) need $\nabla C(\boldsymbol{\theta})$. On a classical computer with a differentiable cost, you get this for free via automatic differentiation. On a quantum computer, autodiff does not work: the "forward pass" is a physical quantum circuit, not a Python trace.

So you are stuck estimating $\partial C/\partial\theta_i$ from black-box evaluations of C. There are two competing strategies.

The Bad Strategy: Finite Differences

The classical numerical-analysis move: pick a small h, compute

$$\frac{\partial C}{\partial \theta_i} \approx \frac{C(\boldsymbol{\theta} + h\mathbf{e}_i) - C(\boldsymbol{\theta} - h\mathbf{e}_i)}{2h}.$$

This is bad for VQA for two reasons.

First, **bias vs variance tradeoff.** Small h reduces the truncation error but amplifies shot noise: the numerator is a small difference of two noisy quantities, and the variance of the estimator scales as $1/h^2$. Large h reduces noise amplification but increases truncation bias. You cannot win both.

Second, **it does not have to be approximate.** VQAs have a structural property — periodicity of Pauli-rotation gates — that allows gradients to be computed **exactly** (up to shot noise) with two function evaluations. The parameter-shift rule.

The Good Strategy: The Parameter-Shift Rule

Claim. If the cost function has the form

$$C(\boldsymbol{\theta}) = \langle 0|U^\dagger(\boldsymbol{\theta})HU(\boldsymbol{\theta})|0\rangle,$$

and the parameter θ_i enters through a single-qubit Pauli rotation $R_P(\theta_i) = \exp(-i\theta_i P/2)$, then

$$\frac{\partial C}{\partial \theta_i} = \frac{1}{2}\Big[C\big(\boldsymbol{\theta} + \tfrac{\pi}{2}\mathbf{e}_i\big) - C\big(\boldsymbol{\theta} - \tfrac{\pi}{2}\mathbf{e}_i\big)\Big].$$

This is exact. No Taylor expansion, no small-h limit. The only noise in the estimate comes from shot noise in the two function evaluations, not from truncation.

Why It Works — The Derivation Sketch

For a Pauli rotation $R_P(\theta) = \cos(\theta/2)I - i\sin(\theta/2)P$ (recall $P^2 = I$), a direct calculation gives

$$\frac{d}{d\theta}R_P(\theta) = -\frac{i}{2}P \cdot R_P(\theta).$$

Plugging into $C(\theta) = \langle 0|R_P^\dagger(\theta)AR_P(\theta)|0\rangle$ (for some effective observable A that absorbs everything else) and using the commutator identity $[P, A] = \frac{1}{i}\big(R_P(\pi/2)AR_P(-\pi/2) - R_P(-\pi/2)AR_P(\pi/2)\big)$, the derivative collapses to a difference of two cost evaluations at shifted angles.

The full derivation is in Mitarai et al. 2018 and Schuld et al. 2019. The key ingredient is that P has only two distinct eigenvalues (± 1), which constrains the Fourier decomposition of $C(\theta)$ to contain only the two frequencies ± 1. The shift rule is effectively a **trigonometric interpolation identity** evaluated at sample points that the structure of Pauli rotations makes exact.

The Cost

Two circuit evaluations per parameter. For a circuit with P parameters, one full gradient costs $2P$ evaluations. Each evaluation itself costs M measurement circuits (one per commuting Pauli group) times N shots.

So the per-gradient shot budget is roughly $2PMN$. For a 40-parameter circuit, 50 Pauli terms, 1024 shots: **4.1 million shots per gradient**. This is why shot efficiency matters.

Alternatives When Parameter-Shift Is Too Expensive

SPSA (Simultaneous Perturbation Stochastic Approximation). Estimates the whole gradient in two circuit evaluations, independent of P. The catch: the estimate is high-variance, biased against individual components, and works only in expectation. But for very high-dimensional circuits where $2P$ evaluations is prohibitive, SPSA is the canonical choice.

Gradient-free methods. COBYLA, Nelder-Mead, and similar methods avoid gradients entirely. They scale poorly with P but are robust to noise. Often competitive for small circuits.

Natural gradient. Uses a Fisher-information-weighted gradient direction. Requires extra circuit evaluations to estimate the quantum Fisher information matrix (see Module 6), but converges faster per iteration in many problem classes.

Structural Role

This node is where the **parameter-shift rule** enters the course, and it is a pivotal result. It is what makes gradient-based optimization viable for VQA at all. Every time node 1.5 or later refers to "a gradient step," the gradient is being estimated by the shift rule under the hood.

It is also the technical detail that node 1.6 (VQA as system identification) depends on: the shift rule gives the optimizer a noisy but unbiased gradient oracle, which is exactly what first-order sysid methods need.

Relations to Other Concepts

- **Automatic differentiation** — the classical-ML tool the shift rule is trying to replace.
- **Finite-difference methods** — the naive alternative. Worse variance, worse bias, same cost.

- **Simultaneous Perturbation Stochastic Approximation (SPSA)** — Spall 1992; independent of parameter count.
- **Hadamard test** — another quantum primitive for estimating derivatives, via controlled gates. Higher circuit-depth overhead than the shift rule.
- **Quantum natural gradient** — Stokes et al. 2020; uses Fubini-Study metric.

Worked Example — Parameter-Shift on a One-Qubit RY Circuit

Consider

$$|\psi(\theta)\rangle = R_Y(\theta)|0\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle.$$

With $H = Z$, the cost is $C(\theta) = \cos\theta$ and the exact gradient is $-\sin\theta$.

Apply the shift rule:

$$\frac{\partial C}{\partial \theta} \stackrel{?}{=} \frac{1}{2}\big[\cos(\theta + \pi/2) - \cos(\theta - \pi/2)\big] = \frac{1}{2}\big[-\sin\theta - \sin\theta\big] = -\sin\theta. \checkmark$$

Exact, not approximate. Every time you use the shift rule in VQE or QAOA, this is what is happening under the hood.

Visualization

*Pending. Diagram: cost function $C(\theta)$ plotted over one period. Two evaluation points shown at $\theta \pm \pi/2$. Arrow between them, labeled with the shift-rule formula. Annotate that the slope of the chord between these two points **equals** the tangent slope at θ — exactly.*

Exercises

1. Verify the parameter-shift rule for $R_X(\theta)$ acting on $|0\rangle$ with observable $H = Z$. Compute both sides of the identity and show they agree.
2. Derive the variance of the shift-rule gradient estimator assuming each evaluation has shot variance σ^2. Compare to the variance of a central finite difference with step h.
3. (VQA) For a Hamiltonian with $M = 100$ Pauli terms and a circuit with $P = 20$ parameters, estimate the shot cost of one full parameter-shift gradient at 1024 shots per measurement. Compare to SPSA.

Research Extensions

- **Higher-order parameter shifts** — for gates whose generators have more than two distinct eigenvalues, the shift rule generalizes to more than two evaluation points (Wierichs et al. 2022).
- **Stochastic parameter-shift** — sampling parameters to estimate gradients; trades bias for variance.
- **Differentiable quantum circuits** — autodiff frameworks (PennyLane, TorchQuantum) that compose shift-rule gradients with classical autodiff.

Cross-links to Other Courses

- **Calculus / Numerical Analysis** — finite differences, truncation error, bias-variance tradeoffs.
- **Cost Landscapes (Module 3)** — gradient information as a probe of landscape geometry.
- **Optimization Dynamics (Module 6)** — how gradient estimators interact with update rules.

The Hybrid Optimization Landscape

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.5

This is the **"what does the classical optimizer actually see?"** node. Having established the architecture (1.2), how expectations are measured (1.3), and how gradients are estimated (1.4), this node steps inside the classical layer and describes the optimization problem from its point of view.

The answer is uncomfortable: the optimizer is trying to minimize a function that is noisy, bounded, non-convex, periodic, and only accessible through an expensive oracle. Classical optimization textbooks do not prepare you for all of those at once.

The Five Defining Features

1. Noisy

Every call to the cost oracle returns a random variable, not a number. Shot noise from finite measurement (node 1.3) is intrinsic. Hardware noise on real devices compounds it. The optimizer cannot "turn off" the noise — only budget against it.

Consequence: algorithms that assume deterministic function values (Newton's method, BFGS) either fail outright or need substantial modification.

2. Periodic

Parameters are angles. For any Pauli rotation gate $R_P(\theta) = \exp(-i\theta P/2)$ with eigenvalues ± 1,

$$C(\boldsymbol{\theta} + 2\pi \mathbf{e}_i) = C(\boldsymbol{\theta}).$$

The natural domain is a torus $[-\pi, \pi]^P$, not $\mathbb{R}^P$.

Consequence: "going to infinity" is not a mode of failure; the optimizer cannot leave the compact domain. Also consequence: standard L2 regularization (as a prior-around-origin) is conceptually awkward here (see node 5.1).

3. Non-convex

The cost $C(\boldsymbol{\theta})$ is a sum of expectation values of Pauli strings in a state that depends nonlinearly on $\boldsymbol{\theta}$ through the circuit. It generically has many local minima, saddle points, and flat regions.

Consequence: no global-optimum guarantees from gradient descent. "Convergence" means "the iterates settle into a basin," not "the iterates reach the minimum."

4. Potentially Flat (Barren Plateaus)

For random parameter initializations in expressive deep circuits, $\nabla C(\boldsymbol{\theta})$ is exponentially small in the number of qubits — this is the barren plateau phenomenon (Module 4). Large swathes of parameter space look perfectly flat to a local optimizer.

Consequence: the optimizer can spend arbitrarily many shots in a region with no detectable signal. Global exploration or warm-started initialization is often necessary.

5. Manifold-Constrained

The states $\{|\psi(\boldsymbol{\theta})\rangle\}$ reachable by the ansatz are a restricted submanifold of Hilbert space. The optimizer never directly sees this manifold; it sees only a P-dimensional parameter space. But the structure of the optimization problem — its curvature, its Hessian, its natural metric — is inherited from the manifold, not from Euclidean parameter space.

Consequence: the "natural gradient" (Module 6) can outperform vanilla gradient descent dramatically, because it respects the manifold geometry rather than the parameter coordinate chart.

What This Looks Like In Code

Imagine you are an SGD optimizer running in a VQA. At each step:

1. You request a gradient.
2. You wait several hundred milliseconds to several seconds for the quantum hardware to return an answer.
3. The answer is a vector whose components are the true gradient plus large random noise.
4. You take a step of size η.
5. The step might have been a descent direction; it might have been sideways; it might have increased the cost. You cannot tell from a single measurement.
6. Repeat a few thousand times and hope it works out.

This is not "noisy classical optimization with a small correction for uncertainty." This is a qualitatively different setting, and the algorithms that work best here (HOPSO, SPSA, carefully regularized gradient methods) are not the algorithms that dominate classical ML.

Structural Role

This node is the **motivation** for the rest of the course. Every topic downstream of Module 1 is in some sense a response to one or more of the five features above:

- Module 3 (Cost Landscapes) studies the geometry — features 3, 4, 5.
- Module 4 (Barren Plateaus) is feature 4, in full.
- Module 5 (Regularization) is a set of techniques for handling feature 1 without pretending it isn't there.
- Module 6 (Optimization Dynamics) designs update rules appropriate for features 1 and 5.
- Module 7 (Control Theory View) formalizes the whole thing as a disturbance-rejection problem.
- Module 9 (Hardware Noise) is the physical origin story of feature 1.

If you want to know "why do we need so many modules just to optimize a function," this node is why.

Relations to Other Concepts

- **Stochastic optimization** — the broad classical umbrella; Robbins-Monro, Kiefer-Wolfowitz, SGD.
- **Bandit problems** — each parameter-space query is like pulling an arm; the optimizer has a limited query budget.
- **Bayesian optimization** — an alternative classical strategy for noisy black-box optimization; works but has its own scaling issues.
- **Non-convex optimization theory** — the parts that don't assume smoothness or convexity; recent results on overparameterized neural networks sometimes transfer.

Worked Example — The Optimizer's Uncertainty Budget

Suppose a single circuit evaluation returns $\widehat{C}(\boldsymbol{\theta}) = C(\boldsymbol{\theta}) + \varepsilon$ with $\mathrm{Var}[\varepsilon] = \sigma^2$. A gradient component via the parameter-shift rule is

$$\widehat{g}_i = \frac{1}{2}\big[\widehat{C}(\boldsymbol{\theta} + \tfrac{\pi}{2}\mathbf{e}_i) - \widehat{C}(\boldsymbol{\theta} - \tfrac{\pi}{2}\mathbf{e}_i)\big],$$

with variance $\sigma^2/2$ (two independent noisy evaluations, divided by 2, squared). If $\sigma = 0.1$, then the gradient estimate has a per-component standard deviation of about 0.07.

For the optimizer to reliably distinguish "gradient pointing left" from "gradient pointing right," the true gradient magnitude must exceed this noise floor. In a barren-plateau regime where the true gradient is $O(2^{-n})$, this requires exponentially many shots just to resolve the sign. This is the quantitative statement of "feature 4 eats your shot budget."

Visualization

Pending. Three-panel diagram: 1. *Periodic 2D cost surface on a torus, annotated "periodic, non-convex."* 2. *Same surface with random noise overlaid on every evaluation, annotated "shot-noisy."* 3. *Same with a large flat region highlighted, annotated "barren plateau." One image, one caption: "This is what the optimizer is working with."*

Exercises

1. For a two-parameter toy cost $C(\theta_1, \theta_2) = \cos\theta_1 \cos\theta_2$, enumerate all critical points and classify them. How many are global minima, how many local minima, how many saddles?
2. Show that adding independent Gaussian noise to each gradient component preserves the expected descent direction but can arbitrarily delay convergence for small true gradients. Quantify the delay in terms of signal-to-noise ratio.
3. (VQA) Give a qualitative argument for why periodicity of the parameter space *helps* an optimizer that gets stuck: what escape routes does the torus geometry offer that $\mathbb{R}^P$ does not?

Research Extensions

- **Landscape smoothness characterization** — how does $\|\nabla^2 C\|_\infty$ scale with depth and qubit count?
- **Warm starting** — classical pre-optimization on a cheaper surrogate to get out of barren regions before spending quantum resources.
- **Adversarial landscapes** — worst-case Hamiltonians designed to fool specific optimizers.

Cross-links to Other Courses

- **Optimization Theory** — stochastic and non-convex chapters.
- **Barren Plateaus (Module 4)** — feature 4 in full.
- **Cost Landscapes (Module 3)** — features 3, 5 in full.
- **Statistics** — noise models, estimator variance.

VQA as System Identification

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.6

This node offers a reframing: **variational quantum algorithms are a special case of system identification.** The classical optimizer is trying to learn the shape of an unknown function through noisy probes, and the structure of that problem is the same whether the function happens to be a physics Hamiltonian expectation, an industrial plant's response curve, or a simulator's reward landscape.

This framing is not just cosmetic. It lets you import a century of control-theory and sysid literature into VQA, and it foreshadows Module 7 (control-theoretic view).

System Identification in One Paragraph

System identification is the classical engineering discipline of building a model of an unknown dynamical system by observing its input-output behavior. You apply known inputs, measure the outputs, and infer a model — either a parametric model (fit the parameters of a chosen structure) or a non-parametric one (estimate the response function directly).

The three defining features of a sysid problem are:

1. You have **query access** but not **introspection access**. You can poke the system and see what it does, but you cannot look inside.
2. Observations are **noisy**. Every measurement carries sensor or sampling error.
3. The **query budget is finite**. You cannot run experiments forever.

Every one of these bullets applies to VQA verbatim if you replace "unknown dynamical system" with "the landscape $C(\boldsymbol{\theta})$ generated by a quantum circuit."

The VQA-Sysid Map

Sysid concept	VQA instance
Unknown system	The cost landscape $C(\boldsymbol{\theta})$ generated by quantum circuit + Hamiltonian
Input	Parameter vector $\boldsymbol{\theta}_k$
Output	Noisy scalar $\widehat{C}(\boldsymbol{\theta}_k)$
Sensor noise	Shot noise + hardware errors
Query budget	Shot budget
Model class	Implicit in the optimizer's update rule
Identification goal	Find $\arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$

The last row is where VQA differs slightly from classical sysid: in most sysid applications the goal is to build an accurate model for downstream use (prediction, control), whereas in VQA the goal is to find an optimal input without necessarily caring whether you have learned the full landscape.

This makes VQA a special case of **optimization-driven system identification** — a subfield of sysid where you only care about model accuracy near the optimum. Bayesian optimization is the classical archetype.

Why This Reframing Is Useful

It makes the right quantities the right quantities. A sysid practitioner intuitively knows that the relevant cost metric is "queries used to reach a given accuracy," not "iterations of my inner loop." For VQA this maps to "shots used to reach a given energy accuracy." Getting this x-axis right (see node 5.9, metric 4) is a common place where VQA benchmarks go wrong.

It tells you when classical intuition fails. Most sysid literature assumes linear, time-invariant, or at most smoothly nonlinear systems. VQA landscapes are none of those. But the *framework* of sysid — query, observe, update belief, query again — transfers cleanly. You just need nonlinear/non-parametric identification tools.

It connects to control theory. Sysid is the data-acquisition half of the sysid-plus-control pipeline. Module 7 picks up the other half: once you have identified enough of the landscape to act, how do you design update rules that steer the system effectively? The control framing makes regularization, stability analysis, and disturbance rejection (Module 5) look like standard techniques rather than ad-hoc tricks.

It motivates query-efficient algorithms. In sysid, you care about sample complexity — how many queries are needed in the worst case. This is exactly the shot budget question for VQA. Methods from bandit theory (UCB, Thompson sampling), Bayesian optimization (Gaussian process surrogates), and active learning are all candidates — and many of them are being actively studied as VQA optimizers.

What VQA Is Not

It is worth naming what this framing does **not** say.

- VQA is not **identifying the Hamiltonian**. The Hamiltonian is known; it is an input to the problem. VQA is identifying the response surface $C(\boldsymbol{\theta})$ that the known Hamiltonian induces through a known ansatz.
- VQA is not **quantum-state tomography**. Tomography would mean reconstructing the full state $|\psi(\boldsymbol{\theta})\rangle$ at each $\boldsymbol{\theta}$, which is exponentially expensive. VQA collapses the state to a single number at each query.
- VQA is not **learning a quantum channel**. The channel (the circuit) is fixed and known. VQA is just searching over its input parameters.

The "unknown" in VQA's sysid interpretation is only the **function** from parameters to expectation values — its local minima, its curvature, where to step next. Nothing quantum remains unknown from the optimizer's perspective. This is the same observation that node 1.7 makes in different language.

Structural Role

This node is the **conceptual bridge** from Module 1 to Module 7 (Control-Theoretic View). It introduces the sysid framing lightly, without the full machinery. Module 7 will formalize it: observability, controllability, the Pontryagin principle applied to VQA trajectories.

It also sets up node 1.9 (separation of objective and dynamics), which is the thesis-flavor version of the same decomposition: the "system" (quantum substrate) and the "identifier" (classical optimizer) are distinct, and you can study them separately before putting them back together.

Relations to Other Concepts

- **Bayesian optimization** — builds an explicit surrogate (Gaussian process or similar) of the unknown function and uses it to decide where to query next.

- **Active learning** — choosing queries to maximize information gain; relevant to shot allocation.
- **Multi-armed bandits** — each parameter location is an arm with unknown reward distribution.
- **Reinforcement learning** — policy-gradient methods are structurally similar; see Module 10.
- **Gaussian process regression** — a common sysid surrogate model; used in Bayesian optimization for VQA.

Worked Example — A Surrogate-Model Optimizer

Consider this simple VQA-sysid algorithm:

1. Sample K random parameter points $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(K)}$.
2. Evaluate $\widehat{C}(\boldsymbol{\theta}^{(i)})$ for each.
3. Fit a Gaussian process surrogate $\widetilde{C}(\boldsymbol{\theta})$ with estimated mean and variance.
4. Choose the next query as $\boldsymbol{\theta}^{(K+1)} = \arg\min_\theta[\widetilde{C}(\boldsymbol{\theta}) - \kappa\sigma(\boldsymbol{\theta})]$ — the lower confidence bound.
5. Evaluate, add to the training set, refit.
6. Repeat.

This is Bayesian optimization applied to VQA. It is pure sysid: the GP is an explicit model of the unknown landscape, and queries are chosen to trade off exploration (high σ) against exploitation (low $\widetilde{C}$). It works surprisingly well on small VQE instances and is a live research direction.

Visualization

Pending. Diagram: a black box labeled "quantum device" with an input arrow $\boldsymbol{\theta}$ and an output arrow $\widehat{C}(\boldsymbol{\theta})$. Around it, a classical layer that maintains a "belief" about the shape of $C(\boldsymbol{\theta})$ (shown as a dotted contour map) and updates the belief with each query. Arrows feeding back to choose the next query.

Exercises

1. Given a fixed shot budget B, is it better to use all shots for one very accurate evaluation or split them across many noisier evaluations? Argue both sides for an exploration phase and an exploitation phase.
2. Describe a VQA optimizer that would be natural from a pure bandit perspective. What is the reward, what is the regret metric, and what assumption lets you bound regret?
3. (VQA) Why is Bayesian optimization's $O(K^3)$ classical cost per iteration a concern for large-parameter VQAs? What is the tradeoff between quantum shot cost and classical surrogate cost?

Research Extensions

- **Quantum kernel methods** — using quantum circuits as feature maps inside classical sysid pipelines.
- **Information-theoretic shot allocation** — querying where the surrogate's uncertainty is highest, weighted by expected reduction in posterior variance.
- **Meta-learning over problem families** — learning a good warm-start distribution across many VQE instances.

Cross-links to Other Courses

- **Control-Theoretic View (Module 7)** — the natural next step.
- **Statistics Module 6** — Bayesian inference and Gaussian process regression.

- **Game Theory / Bandits** — exploration-exploitation tradeoffs.
- **Systems Thinking** — sysid as a general discipline, not just a VQA trick.

Epistemic vs Ontological Uncertainty: The Poker/Bell Frame

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.7

This node is the **authored conceptual lens** of the VQA course. It offers a distinction — between two fundamentally different kinds of uncertainty — and then claims that the classical-quantum interface in VQA sits on the boundary between them, in a very specific way.

This framing is original to this course. It is not a theorem, not a benchmark, not a new algorithm. It is a **way of seeing** the subject that makes several otherwise-confusing decisions feel obvious. If you internalize it, a lot of the rest of the course stops being surprising.

Two Kinds of Uncertainty

Compare two games.

Poker

You are playing Texas Hold'em. Your opponent has two cards face down. You do not know what they are. But **there is a fact of the matter**: the cards are what they are. If the table collapsed and the cards flipped up, you would observe a definite value. Your uncertainty is a statement about **your knowledge**, not about the cards. It is epistemic.

Formally: there exists a hidden state s and a probability distribution $p(s)$ over what that state might be. Observation reveals s and collapses your distribution to a delta. The distribution was never a physical thing; it was a bookkeeping device for your ignorance.

Bell State

You hold one qubit of a pair in the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. You are asked: what will you measure in the Z basis? The answer is: 50% chance of 0, 50% chance of 1. But — and this is the content of Bell's theorem — **there is no fact of the matter before you measure**. The qubit does not secretly "have" a definite Z value that you are ignorant of. It is in a genuine superposition, and the outcome is generated in the act of measurement.

Formally: there is no hidden state s that would reproduce quantum statistics in all possible measurement bases while respecting locality. Your uncertainty is not a statement about your knowledge; it is a statement about **physical reality itself**. It is ontological.

Why The Distinction Matters

In **classical** probability, epistemic and ontological collapse into each other and there is no need to distinguish them operationally. If you are flipping a fair coin, it is equivalent to say "there is a 50% chance it's heads" whether you think the coin is physically undetermined until it lands or that determinism rules the trajectory and you just don't know the initial conditions. Either interpretation yields the same bookkeeping.

In **quantum** mechanics, the distinction is physically real. Bell inequalities can be violated; no-go theorems exist; experiments confirm that quantum uncertainty is not reducible to hidden variables (modulo interpretational sophistries that no one makes practical decisions on). The uncertainty in quantum measurement outcomes is ontologically different from the uncertainty in a face-down poker card.

The VQA Twist: The Optimizer Lives in Poker World

Here is the authored claim.

A variational quantum algorithm's quantum substrate is ontologically quantum — it genuinely prepares superpositions, entanglement, and measurement outcomes that cannot be simulated by any hidden-variable model. It lives in Bell-state world.

But the **classical optimizer** never sees any of that. The optimizer receives scalar numbers $\widehat{C}(\boldsymbol{\theta}_k)$. From its point of view, those numbers are **samples** from a distribution, and the distribution's mean is the **true** value $C(\boldsymbol{\theta}_k)$. The "true" value is a well-defined real number — it exists, it has a fact of the matter, it is what the sample mean converges to as shot count goes to infinity.

From the optimizer's vantage point, $C(\boldsymbol{\theta})$ is a deterministic function of $\boldsymbol{\theta}$ corrupted by sampling noise. This is exactly the structure of poker — there is a fact, we just haven't sampled enough to see it clearly.

The epistemic / ontological distinction has been **collapsed at the measurement interface**. Everything Bell-state about the quantum substrate has been converted, by the Born rule and shot averaging, into Poker-style epistemic uncertainty about a well-defined scalar.

Why This Framing Unlocks Things

Classical intuition is (almost) safe. The optimizer can safely reason about $C(\boldsymbol{\theta})$ as if it were a classical noisy function. Concepts from classical stochastic optimization — variance reduction, gradient descent, confidence intervals, the Chernoff bound — all apply. The optimizer does not need to know or care about the quantum origin of the noise.

It's why optimizer-agnostic analysis works. Because the interface exposes a poker-world function, you can swap optimizers freely without worrying about whether they "understand quantum mechanics." They don't need to; the thing they are optimizing is, from their perspective, a classical stochastic oracle. This is the foundation for the thesis's optimizer-agnostic treatment of regularization (Module 5) and population methods (Modules 4-5 of the thesis).

It tells you where "quantum advantage" can and cannot help. The quantum substrate gives you access to a specific family of poker-world functions $C(\boldsymbol{\theta})$ that are expensive or impossible to evaluate classically. The quantum advantage is entirely in **the menu of functions you can query**, not in **how you query them**. Every optimizer strategy that beats a classical baseline does so because it exploits structure in this menu, not because it does something "quantum."

It demystifies noise models. Different hardware noise profiles give you different poker-world functions (with different variance, different bias, different correlations). They do not give you a "more quantum" or "less quantum" problem. Noise-mitigation techniques are classical variance-reduction strategies applied to a classical statistical problem — which is why they can be analyzed with classical tools.

Where The Frame Breaks

The poker/Bell collapse happens **at the interface**. Before measurement, the quantum state is genuinely Bell-state. The variance of the cost estimator depends on the full quantum state's properties: $\mathrm{Var}[\widehat{C}(\boldsymbol{\theta})]$ depends on $\langle\psi|H^2|\psi\rangle - \langle\psi|H|\psi\rangle^2$, which is a genuinely quantum quantity. Barren plateaus (Module 4) are a property of the ontologically quantum state ensemble, even though they show up to the optimizer as "my gradients are suspiciously small."

So the frame is: **ontological quantum physics determines the shape and noise structure of a poker-world function**, which the optimizer then treats as epistemic classical uncertainty. Every "surprising"

feature of VQA optimization comes from a place where the frame leaks — where the quantum origin of the noise structure shows up in the classical problem.

This is why Module 9 (hardware noise) is important and why you can't just pretend VQA is classical optimization with a noise model pulled out of a hat.

Structural Role

This is the **authored lens** of Module 1 and the conceptual anchor for the whole course's treatment of the classical-quantum interface. It sits between node 1.6 (sysid view) and node 1.8 (thesis-flavor objective generator) because it provides the philosophical bridge between the two.

Where node 1.6 is a classical-engineering framing and node 1.8 is a thesis-vocabulary framing, this node is the underlying **why** that makes both of them legitimate.

Relations to Other Concepts

- **Frequentist vs Bayesian probability** — a related but distinct divide; orthogonal to epistemic vs ontological.
- **Hidden variable theories** — the historical attempt to reduce quantum uncertainty to classical ignorance. Bell's theorem rules out the local version.
- **QBism** — a quantum interpretation that reframes quantum probabilities as agent-epistemic rather than ontological. Not the same as the claim here; QBism is about the *physics*, the claim here is about the *interface*.
- **Decoherence** — the physical mechanism by which ontologically quantum states develop effectively classical statistics.

Worked Example — A Single Pauli Expectation

Let $H = Z$ and $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$. True value: $C(\theta) = \cos\theta$. Measured value over N shots: a sample mean $\widehat{C}(\theta)$ with variance $\sin^2\theta/N$.

Bell-state view: each shot is a genuinely quantum measurement; the outcome does not exist until the measurement happens; the ± 1 results are ontologically non-pre-existing.

Poker view (optimizer's): there is a number, $\cos\theta$, and I have a sample mean that approximates it. The sample mean has variance $\sin^2\theta/N$. I want to use the sample mean to decide how to adjust θ.

Both views are **simultaneously correct**. The Bell-state view describes what is physically happening. The poker view describes what the optimizer sees. They agree on the mean and the variance. The optimizer's job is easier because it can ignore everything in the Bell-state view except the statistical shape of the output.

Visualization

Pending. Two-panel diagram. Left: a poker table with two face-down cards, labeled "EPISTEMIC: there is a fact, I don't know it." Right: a Bell pair with an entanglement swirl, labeled "ONTOLOGICAL: there is no fact until measurement." Below both, a horizontal arrow pointing to a single box labeled "$\widehat{C}(\boldsymbol{\theta}) \in \mathbb{R}$ — what the optimizer sees" with caption: "The Born rule + shot averaging collapses the right panel into the left panel at the interface."

Exercises

1. For a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, explain the difference between saying "there is a 50% chance of measuring 0" and "the qubit has a definite value of 0 half the time." Which is consistent with Bell's theorem?
2. The sample mean $\widehat{C}(\theta)$ converges to $\cos\theta$ in the infinite-shot limit. Is $\cos\theta$ an epistemic or ontological quantity in the framing of this node? Defend your answer.
3. (VQA) Give an example of a feature of VQA optimization where the epistemic-poker framing is **insufficient** and the ontologically-quantum origin of the problem matters. (Hint: Module 4.)

Research Extensions

- **Interface-collapse theorems** — can this poker/Bell collapse be made quantitatively precise? What exactly is lost at the interface?
- **Quantum-native optimizers** — optimizers that act on the state itself (via swap tests, amplitude estimation, or partial tomography) rather than on scalar cost feedback. These genuinely operate on the Bell-state side and may access information the poker-framed optimizer cannot.
- **Philosophy-of-VQA** — cross-referencing this framing with the broader quantum interpretations literature.

Cross-links to Other Courses

- **Quantum Foundations** — Bell's theorem, hidden variables, interpretations.
- **Statistics Module 1** — frequentist and Bayesian probability frameworks.
- **Game Theory** — poker as the canonical epistemic-uncertainty game.
- **Systems Thinking** — the general pattern of complicated internals presenting a simpler interface.

The Quantum Subsystem as Stochastic Objective Generator

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.8

This node states the **thesis-vocabulary** version of what the quantum layer does. Nodes 1.2 (architecture) and 1.3 (expectation estimation) described the mechanics. Node 1.7 (poker/Bell) described the philosophy. This node collapses both into the single working abstraction used throughout the rest of the course: **the quantum device is a stochastic objective generator**, and nothing more.

It is a deliberately **minimalist** abstraction. Its purpose is to let the classical optimizer ignore almost everything quantum and focus on what it can actually respond to.

The Definition

A **stochastic objective generator** is an oracle with the following interface:

- **Input:** a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^P$.
- **Output:** a real number $\widehat{C}(\boldsymbol{\theta}) \in \mathbb{R}$, drawn from a distribution whose mean is a fixed (but oracle-hidden) function $C(\boldsymbol{\theta})$ and whose variance is bounded.
- **Stateless:** successive calls at the same $\boldsymbol{\theta}$ return independent samples; there is no internal memory or adaptivity.
- **Opaque:** the internal mechanism producing $\widehat{C}(\boldsymbol{\theta})$ is not inspectable from the outside. The caller has query access only.

That is the entire signature. Nothing in this description mentions Hilbert spaces, amplitudes, entanglement, measurement, qubits, or Pauli strings.

Claim: the quantum subsystem of a VQA is exactly a stochastic objective generator in this sense.

This is not a metaphor and not an approximation. It is a literal description of the interface the classical optimizer interacts with.

What Is Gained By This Abstraction

Optimizer-agnostic analysis. Any result you prove about how optimizers interact with stochastic objective generators applies to VQAs, regardless of which quantum hardware, which ansatz, which Hamiltonian. This is why Module 5's regularization thesis can be stated without ever invoking specific quantum details: the argument is about the oracle interface, not about what's on the other side.

Clean separation of concerns. The quantum-hardware researcher's job is to make the oracle cheaper, lower-variance, and more reliable. The optimizer researcher's job is to make the best use of the oracle interface. These two jobs can proceed in parallel and exchange results through the shared abstraction.

Clear failure diagnostics. When a VQA fails, you can ask: is the oracle broken (variance too high, bias too large, correlated errors) or is the optimizer broken (bad update rule given the observed variance profile)? Without the abstraction, these two failure modes are indistinguishable; with it, they are separable.

Composability with classical tools. Any classical tool that accepts a stochastic oracle — Bayesian optimization, bandit algorithms, simulation-based inference, evolutionary strategies — can be plugged in without reinvention. The oracle doesn't care that it's quantum; the classical tool doesn't care that the oracle is quantum. They meet at the interface and shake hands.

What Is Lost By This Abstraction

Quantum structure of the noise. The variance of $\widehat{C}(\boldsymbol{\theta})$ depends on quantum quantities (e.g., $\langle\psi|H^2|\psi\rangle - \langle\psi|H|\psi\rangle^2$) that are not exposed through the scalar interface. An optimizer that only sees samples cannot in general infer this variance without extra queries.

Structural exploitability. Some potential optimization strategies might exploit the fact that the oracle **is** a quantum circuit — for example, by requesting measurements in alternative bases, by running swap tests, by using amplitude estimation. Those are genuinely quantum-enhanced strategies, and they live **outside** the stochastic-objective-generator abstraction. Treating the quantum side as an opaque oracle means you give up on these.

Circuit-level insight. Phenomena like barren plateaus depend on the global structure of the parameterized circuit. A pure-oracle view sees them only as "gradients are unexpectedly small"; diagnosing the root cause requires cracking the oracle open.

This is an acceptable tradeoff for Module 1's purposes — the abstraction is a working tool, not a permanent wall. Later modules (especially 4, 9, 10) crack the oracle open when they need to.

Relationship to Node 1.7 (Epistemic vs Ontological)

Node 1.7 gave the philosophical justification; this node gives the operational consequence.

Node 1.7 argued that the quantum-classical interface collapses Bell-state ontological uncertainty into poker-style epistemic uncertainty about a well-defined scalar. This node takes that collapse at face value and **builds an abstraction around it**. The stochastic objective generator is the formal object you get when you take the poker view seriously and treat the resulting classical-looking oracle as the primary thing to reason about.

You could not have one without the other. Without 1.7, the objective-generator abstraction looks like a pragmatic shortcut. With 1.7, it looks like the mathematically correct quotient to take.

Relationship to Node 1.6 (System Identification)

Node 1.6 framed VQA as sysid. A stochastic objective generator is exactly the kind of unknown system that sysid tries to characterize. The stochastic-objective-generator abstraction is therefore the **target** of the sysid framing: you probe a stochastic oracle, you build up a model (explicit or implicit) of its $C(\boldsymbol{\theta})$ surface, you use that model to find the minimum.

Structural Role

This is the last of the "what is the quantum side?" nodes. Nodes 1.2-1.4 described it mechanically. Nodes 1.6-1.7 described it conceptually. This node locks in the minimal working abstraction that Module 5 and beyond will use.

From this point on in the course, whenever you see $\widehat{C}(\boldsymbol{\theta})$ in a diagram or equation, you should mentally substitute: *this is the output of a stochastic objective generator that happens to be implemented by a quantum device.* The abstraction is the thing the math operates on. The implementation is the reason the abstraction is interesting.

Relations to Other Concepts

- **Stochastic oracles** — the general classical computer-science abstraction. Robbins-Monro, Kiefer-Wolfowitz, Polyak, Nemirovski all have results on this.
- **Simulator-based inference** — classical ML with an opaque simulator as the oracle; structurally identical.
- **Black-box optimization** — the general name for optimizing oracles with no derivative access. VQA is a noisy black-box optimization problem.
- **PAC learning with membership queries** — the learning-theory cousin.

Worked Example — Abstracting Away a VQE

A Variational Quantum Eigensolver on H_2 with the UCCSD ansatz has:

- 14 parameters (singles + doubles amplitudes)
- A Hamiltonian with ~15 Pauli terms after Jordan-Wigner transformation
- A fixed number of qubits (4 after tapering)
- Shot noise proportional to $\sim 1/\sqrt{N}$ per Pauli term

From the stochastic-objective-generator perspective, all of that collapses to:

> "A black box that takes $\boldsymbol{\theta} \in [-\pi, \pi]^{14}$ and returns a noisy scalar. Variance structure depends on $\boldsymbol{\theta}$ but is bounded by some known constant. One query costs $\sim 15N$ shots."

An optimizer written against this abstraction will work on H_2, H_2O, N_2, LiH — whatever you point it at — without caring about the physics. This is exactly why HOPSO and SPSA and COBYLA can be benchmarked on "VQE" without specifying the molecule: the oracle abstraction is the level at which the optimizer lives.

Visualization

Pending. Diagram: a black box with input $\boldsymbol{\theta}$, output $\widehat{C}(\boldsymbol{\theta})$. Inside the box, a faint silhouette of a quantum circuit, grayed out to indicate "implementation details." Outside the box, a clean classical optimizer that only sees the input-output pair. Label: "The optimizer sees the box, not the silhouette."

Exercises

1. Give three examples of "stochastic objective generators" from non-quantum contexts. For each, identify the input space, output distribution, and source of noise.
2. The stochastic-objective-generator abstraction throws away information about the variance structure of $\widehat{C}(\boldsymbol{\theta})$. Design an extended interface that preserves the mean **and** variance, and argue whether an optimizer can exploit it. (Hint: think about variance-penalized objectives from node 5.7.)
3. (VQA) Describe a situation where the stochastic-objective-generator abstraction **actively misleads** the optimizer. What is the structure of the failure?

Research Extensions

- **Enriched oracle interfaces** — stochastic generators that also expose variance estimates, cross-query correlations, or gradient samples.
- **Oracle-complexity theory for noisy black-box optimization** — how many queries are needed to find an ε-optimum under bounded variance?

- **Ansatz-agnostic optimizer design** — exploiting only the generator interface, deliberately avoiding circuit-level structure.

Cross-links to Other Courses

- **Statistics Module 6** — stochastic oracles and Monte Carlo estimators.
- **Systems Thinking** — abstraction boundaries and their costs.
- **Game Theory** — stateless opponent models as an analogy.
- **Internet Systems** — APIs as opaque oracles; the same abstraction pattern at a different scale.

The Separation of Objective and Dynamics

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hybrid Quantum-Classical Systems **Node:** 1.9

This is the **closing node** of Module 1 and its most important structural claim.

The thesis of Module 1 compresses to a single sentence: **in a variational quantum algorithm, the objective and the dynamics are two different things, and most confusions in the field come from forgetting that.**

This node is the formal statement of that claim, and the justification for treating "objective" and "dynamics" as the two independent axes along which the rest of the course is organized.

The Two Components

Every VQA has two separable pieces.

The objective. A function $C : \mathbb{R}^P \to \mathbb{R}$, defined implicitly by the choice of ansatz $U(\boldsymbol{\theta})$, Hamiltonian H, and hardware noise model. This function has a shape: local minima, saddle points, flat regions, periodicities. It has a noise profile: the variance of $\widehat{C}(\boldsymbol{\theta})$ as a function of $\boldsymbol{\theta}$ and shot budget. Once the ansatz and Hamiltonian are fixed, the objective is **completely determined** — it exists as a mathematical object independent of any optimizer.

The dynamics. A rule $\mathcal{A}$ for updating $\boldsymbol{\theta}_k \to \boldsymbol{\theta}_{k+1}$ given feedback from $\widehat{C}$. Examples: gradient descent, Adam, HOPSO, COBYLA, SPSA, Bayesian optimization, simulated annealing. The dynamics are **completely determined** by the choice of optimizer and its hyperparameters — they exist as an algorithm independent of any particular objective.

The behavior of the VQA is the interaction of these two things. Not the objective alone. Not the dynamics alone. Their composition.

Why This Separation Is Not Obvious

It should be obvious. It is not. The reason is that in most of the VQA literature, discussions of "why VQA is hard" fold both components into a single narrative and then blame one of them.

Typical examples of the confusion:

- "VQE converges slowly because the landscape has barren plateaus." *Half true.* Barren plateaus are an objective property. But whether slowness becomes failure depends on the dynamics — an optimizer that averages over shots differently can cope; one that takes single-sample gradient steps cannot.
- "Adam doesn't work well for VQA." *Half true.* Adam has particular dynamical properties (momentum, adaptive learning rates per parameter) that interact in particular ways with particular objective shapes. On some objectives Adam is great; on others it is terrible. The statement "Adam doesn't work well for VQA" is the statement "averaged across the VQA objectives the community cares about, Adam's dynamics interact poorly," which is a much narrower claim than the plain reading suggests.
- "Parameterized circuit X is expressive enough to solve problem Y." *Expressivity is an objective property.* But reaching the expressive floor requires dynamics that actually get you there. Expressivity bounds are upper bounds on what is reachable, not statements about what any particular optimizer will find.

The pattern in all of these: conflating a shape-property of the objective with an emergent-property of the optimizer-plus-objective system.

The Formal Statement

Let $\mathcal{O}$ be an objective (cost function plus noise model) and let $\mathcal{A}$ be a dynamics (update rule plus hyperparameters). Define the **behavior** of the pair $(\mathcal{O}, \mathcal{A})$ as the distribution over trajectories $\{\boldsymbol{\theta}_k\}_{k=0}^{T}$ induced by running $\mathcal{A}$ on $\mathcal{O}$ from random initializations.

Claim 1 (decomposition). The behavior is not a function of $\mathcal{O}$ alone or $\mathcal{A}$ alone. It is a function of the pair.

Claim 2 (non-trivial interaction). There exist pairs $(\mathcal{O}_1, \mathcal{A}_1)$ and $(\mathcal{O}_2, \mathcal{A}_2)$ such that swapping dynamics — evaluating $(\mathcal{O}_1, \mathcal{A}_2)$ — produces qualitatively different behavior. In particular, convergence on $(\mathcal{O}_1, \mathcal{A}_1)$ does not imply convergence on $(\mathcal{O}_1, \mathcal{A}_2)$, and vice versa.

Claim 3 (separable study). Despite the non-trivial interaction, one can profitably study the two components separately. Properties of $\mathcal{O}$ (barren plateaus, landscape geometry, concentration phenomena) are well-defined without reference to any optimizer. Properties of $\mathcal{A}$ (stability, convergence rates, noise tolerance) are well-defined without reference to any particular objective. The composition happens last.

Claims 1 and 2 are empirical observations backed by literature (e.g., Arrasmith et al. on optimizer choice in VQA). Claim 3 is the organizational principle of this course.

How The Course Uses This Separation

Every module from here on lives on one side of the divide or the other, or on the composition of the two:

Module	Side
2 — Parameterized Circuits	Objective (shape of the ansatz manifold)
3 — Cost Landscapes	Objective (geometry)
4 — Barren Plateaus	Objective (concentration phenomena)
5 — Regularization	**Composition** — modifies dynamics in response to objective noise
6 — Optimization Dynamics	Dynamics
7 — Control-Theoretic View	Dynamics as controllers acting on objectives
8 — Expressivity vs Trainability	Objective (a shape property, "expressivity", and a composition property, "trainability")
9 — Hardware Noise Models	Objective (modifying the noise profile)
10 — Adaptive Control Systems	**Composition** — closing additional feedback loops

This split is not arbitrary. It follows from the claim of this node. If objective and dynamics were inseparable, each module would need to discuss both, and the course would be an unnavigable tangle. Because they are separable, you can study Module 3 without caring which optimizer will be used, study Module 6 without caring which Hamiltonian is being minimized, and then compose them in Module 5 and beyond.

Connection to the Thesis

Module 5's central claim — regularization as dynamical modification, not prior injection — **depends entirely on this separation**.

The argument goes: prior injection is a statement about the objective (the regularizer rewrites C to a new function $C + \lambda R$). Dynamical modification is a statement about the dynamics (the regularizer changes the

update rule without changing C). These two operations have **different semantics** — they produce different fixed points, different trajectories, different convergence properties.

If objective and dynamics weren't separable, you couldn't distinguish them, and the thesis wouldn't have anything to argue against. The fact that they are separable means there is a real choice to be made about **where** a regularization technique operates, and that choice has consequences.

Node 5.5 makes this argument in full. What this node does is establish the **conceptual substrate** that makes it possible to state the argument at all.

Structural Role

This is the **conceptual capstone** of Module 1. It is also the **organizing principle** of the remaining nine modules. Every time a later module draws a clean line between "landscape" and "optimizer," it is cashing in the separation established here.

If you are reviewing Module 1 and you remember only two things, remember:

1. **Node 1.1:** why minimizing $\langle\psi|H|\psi\rangle$ is a legitimate strategy at all.
2. **Node 1.9:** that the objective and the dynamics are two different things.

Everything else in the module is infrastructure for those two claims.

Relations to Other Concepts

- **No free lunch theorems** — averaged over all possible objectives, all optimizers are equivalent. The separation this node argues for is a prerequisite for stating NFL at all.
- **Plant / controller decomposition** — classical control theory's analogous split. See Module 7.
- **Data / model separation in classical ML** — structurally similar (data defines the loss landscape; architecture + optimizer define the dynamics of fitting it).
- **Game theory** — payoff function (objective) vs strategy (dynamics).

Worked Example — Swapping Optimizers on the Same Landscape

Take a fixed VQE instance: H_2 Hamiltonian, UCCSD ansatz, 14 parameters, 1024 shots per evaluation. Run three optimizers to 500 iterations each:

- **COBYLA** — gradient-free local search. Converges slowly but reliably; low seed variance.
- **Adam with parameter-shift gradients** — gradient-based. Converges faster per iteration but noisier; higher seed variance.
- **HOPSO** — population-based. Different exploration profile; best worst-case performance.

All three experience the **same objective**. Their different behaviors are entirely a property of the dynamics. If you aggregate the results as "VQE on H_2," you hide the fact that the variation across optimizers is larger than the variation across noise seeds — which means **the optimizer choice was the dominant variable**, not the landscape.

This is the empirical signature of Claim 2: dynamics matter, independent of the objective.

Visualization

Pending. Two-axis diagram. Horizontal axis: objective space (a cartoon landscape with hills and valleys). Vertical axis: trajectory space (curves through parameter space). Three different trajectories through the ***same*** *landscape, one per optimizer. Annotation: "same objective, different dynamics, different outcomes." A companion panel: same optimizer on three different landscapes. Annotation: "same dynamics, different objectives, different outcomes."*

Exercises

1. Construct a toy VQA example (2 parameters is enough) where two different optimizers starting from the same initialization reach different local minima. Identify whether the difference is attributable to the objective shape, the dynamics, or the composition.
2. Arrasmith et al. report that different optimizers produce qualitatively different convergence curves on identical VQE instances. Design a benchmark protocol that isolates the dynamics contribution from the objective contribution.
3. (VQA) The "barren plateau" phenomenon is typically described as a property of the objective. But for any fixed objective, some dynamics (e.g., very small step sizes) will still make progress while others (large step sizes) will not. Is "barren plateau" really a pure objective property, or is it a composition property? Defend your answer.

Research Extensions

- **Objective-agnostic optimizer benchmarks** — protocols that report optimizer performance distributions across families of objectives, rather than single instances.
- **Dynamics-agnostic landscape analysis** — geometry, curvature, concentration results that hold regardless of which optimizer will be used.
- **Joint objective-dynamics co-design** — cases where the objective can be modified (via ansatz choice or measurement scheme) to improve the downstream dynamics. HOPSO thesis + regularization thesis together.

Cross-links to Other Courses

- **Control Theory (Module 7 here)** — the plant-controller split.
- **Game Theory** — strategy vs payoff decomposition.
- **Systems Thinking** — the general pattern of "behavior as composition."
- **Statistics Module 10** — model selection as a search over objectives while holding dynamics fixed.

Module 2 — Parameterized Circuits

Ansatz Design

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.1

This is the first node of Module 2 and the **anchor of the quantum half** of the course. Module 1 told you *that* the optimizer interacts with a parameterized state $|\psi(\boldsymbol{\theta})\rangle$. Module 2 starts here by asking the prior question: **what is $U(\boldsymbol{\theta})$, and how is it chosen?**

The answer determines almost everything downstream — what states are reachable, what the cost landscape looks like, whether a barren plateau will appear, and how many shots an optimizer will need. Choosing an ansatz is the single highest-leverage design decision in a VQA.

What an Ansatz Is

An **ansatz** is a parameterized recipe for preparing a family of quantum states. Formally, it is a unitary

$$U(\boldsymbol{\theta}) = \prod_{\ell=1}^{L} U_\ell(\boldsymbol{\theta}_\ell)$$

built from a sequence of layers U_ℓ, each of which depends on a subset of the parameter vector. Acting on a fixed reference state (typically $|0\rangle^{\otimes n}$), the ansatz defines

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})|0\rangle^{\otimes n}.$$

The set of states reachable by varying $\boldsymbol{\theta} \in \mathbb{R}^P$ is the **ansatz manifold** — a strict, structured subset of the full Hilbert space.

The word "ansatz" is borrowed from German theoretical physics and means "starting form" or "trial function." A variational quantum algorithm is, at its core, the act of letting an optimizer slide along this trial form looking for the best parameter assignment.

The Three Design Axes

Every ansatz design choice can be located along three axes.

1. Expressivity — *what states can it reach?* A more expressive ansatz covers more of Hilbert space. At the extreme, a fully universal ansatz can prepare any state, but at the cost of needing exponentially many parameters or exponentially deep circuits. At the other extreme, a 1-parameter ansatz can only prepare a 1-parameter family, regardless of how cleverly you optimize. The interesting region is in between: **just expressive enough to contain a good approximation to the target state, and no more.**

2. Trainability — *can the optimizer find a good point on the manifold?* A maximally expressive ansatz often has barren plateaus (Module 4), meaning the cost gradient vanishes over almost all of parameter space and the optimizer cannot make progress. A less expressive ansatz with the right inductive bias may be much easier to train, even if its theoretical reach is smaller. **Expressivity and trainability are in tension** — Module 8 is dedicated entirely to this tension.

3. Hardware compatibility — *can it run on the device you have?* Real near-term devices have a limited gate set (typically single-qubit rotations + a 2-qubit entangler such as CNOT or CZ), restricted qubit

connectivity (only some pairs of qubits can be entangled directly), and short coherence times (gates must complete before noise destroys the state). An ansatz that is mathematically beautiful but compiles to a circuit that is too deep, requires too many SWAPs, or uses gates the device doesn't support is operationally useless.

A good ansatz design is a Pareto-optimal point on these three axes for the problem you actually care about.

The Two Cultural Camps

VQA ansatz design splits into two communities with very different starting points.

Hardware-first designers (node 2.2): build the ansatz from gates the hardware natively supports, lay them out in a regular pattern (often just "alternating layers of single-qubit rotations and entanglers"), and hope the resulting parameterization is rich enough to contain the answer. This is the **hardware-efficient ansatz**. Simple, agnostic to the problem, runs on anything. Frequently barren-plateau-prone.

Problem-first designers (node 2.3): start from physics or chemistry knowledge, write down a state-preparation recipe motivated by the problem structure (e.g., a unitary coupled-cluster expansion for molecules), then compile that down to hardware gates. This is the **chemically-inspired** or more generally **problem-inspired** ansatz. Better inductive bias, often deeper, often less hardware-friendly.

Both camps are correct about something the other camp is wrong about, and the right answer for any given VQA usually involves elements of both. Modules 2.2 and 2.3 examine each in detail.

What an Ansatz Is Not

It is worth being explicit about what the ansatz does not do.

- **An ansatz does not encode the Hamiltonian.** The Hamiltonian H is the *measurement*, applied to the state the ansatz prepares. The ansatz only prepares the state. (See node 2.5 for how the Hamiltonian is encoded — separately — into a measurement protocol.)
- **An ansatz does not adapt during a single run.** The structure of $U(\boldsymbol{\theta})$ is fixed throughout the optimization; only the parameters $\boldsymbol{\theta}$ change. *Adaptive ansatz methods* (ADAPT-VQE) violate this by growing the ansatz iteratively, but they are a special case treated separately.
- **An ansatz is not unique.** For any target state, infinitely many ansätze can prepare it. The art is choosing one whose parameter landscape is friendly to the optimizer you intend to use.

Structural Role

This node opens Module 2 and sets up everything that follows in it. The next four nodes are direct elaborations of the design axes introduced here:

- 2.2: hardware-efficient ansätze (the hardware-first culture)
- 2.3: chemically-inspired ansätze (the problem-first culture)
- 2.4: compilation, depth, and width — the hardware compatibility axis in detail
- 2.5: Hamiltonian encoding — the *separate* problem of how the Hamiltonian becomes a measurement

The remaining nodes (2.6-2.9) deal with how states are represented, how measurement actually happens on hardware, and the geometric view of the ansatz as a manifold — the deeper structural understanding that pays off when you start reasoning about cost landscapes (Module 3) and barren plateaus (Module 4).

Relations to Other Concepts

- **Variational principle (1.1)** — guarantees that minimizing $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ is a meaningful thing to do regardless of how restricted the ansatz is.
- **Inductive bias in classical ML** — the choice of ansatz is the quantum analogue of the choice of neural network architecture. Both restrict the function class; both inject prior knowledge; both can be over- or under-expressive.
- **Tensor network states** — DMRG, MPS, PEPS — classical analogues of restricted state families with controlled expressivity.
- **Universal quantum computation** — any unitary can in principle be approximated by a sufficiently deep circuit; ansatz design is the art of *not* using a universal one.

Worked Example — Counting the Reachable States

Consider the simplest non-trivial ansatz: a single $R_Y(\theta)$ gate on one qubit, applied to $|0\rangle$.

$$|\psi(\theta)\rangle = R_Y(\theta)|0\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle.$$

The set of reachable states is the **great circle** in the Bloch sphere passing through $|0\rangle$, $|+\rangle$, $|1\rangle$, $|-\rangle$. This is a 1-parameter family — a 1D submanifold of a 2D Hilbert space.

Now add a second layer: $R_Z(\phi)R_Y(\theta)|0\rangle$. The reachable set is now the **entire Bloch sphere** — a 2-parameter family covering all pure single-qubit states. Adding a third parameter $R_Y(\alpha)R_Z(\phi)R_Y(\theta)|0\rangle$ gains nothing in terms of *reachable states* (you can already prepare every single-qubit pure state), but it changes the *parameterization geometry* — the same target state can now be reached via multiple paths, which can either help (more escape routes for an optimizer) or hurt (redundant directions in parameter space).

Lesson: **the number of parameters is not the same as the dimension of the reachable set.** Adding parameters past the point of universality only changes geometry, not coverage — and that geometry change can be either friendly or hostile to optimization.

Visualization

Pending. Three-panel diagram. Panel 1: Bloch sphere with a great circle traced through it (single R_Y ansatz, 1-parameter). Panel 2: full Bloch sphere covered (2-parameter ansatz). Panel 3: same Bloch sphere, but with multiple curves all passing through the same point (3-parameter ansatz, redundant). _Caption: "More parameters can mean more reach, or just more parameterization — these are different._"

Exercises

1. For the 1-parameter ansatz $R_Y(\theta)|0\rangle$, what is the maximum value of $\langle Z\rangle$? What is the minimum? Sketch $\langle Z\rangle(\theta)$ over $\theta \in [-\pi, \pi]$.
2. Show that the ansatz $R_X(\theta_1)R_X(\theta_2)|0\rangle$ is *not* universal for single-qubit states (despite having 2 parameters). What parameter geometry is going wrong?
3. (VQA) Sketch the design tradeoff between expressivity, trainability, and hardware compatibility as a 2D Pareto front for a hypothetical "ideal" VQA ansatz family. Where on the front would you put a hardware-efficient ansatz vs a UCCSD ansatz?

Research Extensions

- **Adaptive ansatz construction (ADAPT-VQE)** — grow the ansatz one operator at a time, choosing the next operator based on its energy gradient. Trades depth for expressivity *only when it helps*.
- **Symmetry-preserving ansätze** — restrict the manifold to states obeying problem symmetries (particle number, total spin, parity), shrinking it to the physically relevant subspace.
- **Learned ansätze** — meta-learn an ansatz structure across families of problems using classical ML pipelines.

Cross-links to Other Courses

- **Machine Learning** — model architecture as inductive bias, capacity vs trainability.
- **Differential Geometry** — submanifolds of complex projective space.
- **Quantum Computing** — universal gate sets, gate decomposition, circuit synthesis.
- **Statistics Module 5** — restricted models and their bias-variance tradeoff.

Hardware-Efficient Ansatz

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.2

The hardware-efficient ansatz (HEA) is the most widely deployed ansatz family in NISQ-era VQAs. It is also the most widely **maligned**, because most barren-plateau theorems were proved against it. Both facts are true, and both are a consequence of the same design philosophy: **build it out of whatever the hardware can do, and let the optimizer figure out the rest.**

The Construction

A hardware-efficient ansatz has three structural ingredients:

1. **A native gate set.** Whatever single-qubit rotations and two-qubit entanglers the device supports natively. Typical: R_X, R_Y, R_Z for single-qubit; CNOT, CZ, or echoed cross-resonance for two-qubit.
2. **A connectivity-respecting layout.** Two-qubit gates only between qubits that are physically adjacent on the device. No virtual SWAPs unless absolutely necessary.
3. **A repeated layer structure.** Stack L identical (up to parameter values) layers, each of the form *single-qubit rotations on every qubit, followed by entanglers between connected pairs.*

A canonical layer of an n-qubit HEA looks like:

$$U_\ell(\boldsymbol{\theta}_\ell) = \left(\prod_{(i,j)\in E} \mathrm{CNOT}_{ij} \right) \cdot \left(\bigotimes_{q=1}^{n} R_Y(\theta^Y_{\ell,q}) R_Z(\theta^Z_{\ell,q}) \right),$$

where E is the set of native qubit pairs. The full ansatz is $U(\boldsymbol{\theta}) = U_L \cdots U_2 U_1$ acting on $|0\rangle^{\otimes n}$.

For n qubits and L layers with two single-qubit parameters per qubit per layer, the parameter count is $P = 2nL$ — linear in both dimensions. This is small enough to be manageable but large enough that for modest n and L, the ansatz becomes expressively rich.

Why It Was Adopted

When IBM, Google, Rigetti, and IonQ first built devices that could run real circuits, the question facing VQA practitioners was practical: **what ansatz can I actually run on this thing tomorrow?**

The HEA answers that question definitively:

- It uses **only native gates**, so no compilation magic is needed.
- It respects **device connectivity**, so no SWAP overhead.
- It is **shallow** (depth $= O(L)$, independent of n), so it fits inside a coherence window.
- It is **agnostic to the problem**, so the same ansatz template works for any Hamiltonian.

The Kandala et al. (2017) paper that introduced the term demonstrated VQE on H_2, LiH, and BeH_2 using this exact recipe and produced energy estimates that, while not chemically accurate, were close enough to demonstrate the approach. That paper became the template for hundreds of follow-up experiments.

Why It Was Maligned

The McClean et al. (2018) barren plateau paper proved that for sufficiently deep, sufficiently random HEAs on sufficiently many qubits, the variance of the cost gradient $\mathrm{Var}[\partial_i C]$ scales as $O(1/4^n)$. This is exponential in the number of qubits.

In practical terms: a randomly initialized HEA on 50 qubits has a gradient so small that no optimizer, no matter how clever, can detect a descent direction without an exponential number of shots. The cost landscape looks **flat** to any local optimizer. The ansatz is mathematically rich enough to contain the answer, but the optimizer cannot find it.

This is the **barren plateau** phenomenon (Module 4 in full), and it is the central technical reason that HEAs are no longer the default recommendation for large VQAs.

The Tension

The HEA's strengths and weaknesses come from the same source.

It is a **universal approximator** in the limit of large L — meaning the manifold it parameterizes covers all of Hilbert space as the number of layers grows. That universality is what makes it general-purpose. It is also what makes the cost landscape look like a uniformly random function on Hilbert space, which is exactly the setting where barren plateaus are guaranteed.

In short: **the HEA is too good at being expressive, and not good enough at being structured.** A problem-inspired ansatz (node 2.3) gives up some expressivity in exchange for inductive bias that lets the optimizer find a good answer in linear rather than exponential queries.

When To Use It Anyway

Despite the criticism, the HEA remains a defensible choice in several settings:

- **Small qubit count** ($n \leq 8$ or so): the barren plateau scaling doesn't bite hard yet, and the HEA's hardware-friendliness wins.
- **Warm-started initialization**: if you have a good initial guess from a classical pre-computation, you can avoid the random-initialization regime where barren plateaus appear, and the HEA may train fine.
- **Symmetry-restricted variants**: an HEA can be modified to preserve problem symmetries (e.g., particle number), which both shrinks the manifold and can mitigate barren plateaus.
- **Benchmarking baseline**: when introducing a new optimization technique, the HEA is the standard baseline ansatz against which improvements are reported. You can't avoid running on it.

The thesis takes an HEA-with-warm-start as one of its evaluation conditions for exactly the reasons above.

Structural Role

This node is one half of the "what kind of ansatz?" pair (the other half is 2.3). It establishes the hardware-first design culture and surfaces the central pathology — barren plateaus — that all subsequent ansatz design tries to escape from. Module 4 returns to this in full when it proves the barren plateau theorem and analyzes its consequences for HEA-based VQAs.

Relations to Other Concepts

- **2-designs and t-designs** — random circuits drawn from sufficiently deep HEAs approximate Haar-random unitaries; the barren plateau theorem uses this to bound gradient variance.
- **Ising-type ansätze** — HEAs whose entangling layers come from time-evolution under an Ising Hamiltonian; structurally similar.
- **Brick-wall circuits** — a specific HEA layout common in tensor network and quantum simulation literature.
- **QAOA (quantum approximate optimization algorithm)** — a different ansatz philosophy where the layers come from time-evolving the problem and mixer Hamiltonians; structurally distinct from HEAs.

Worked Example — A 4-Qubit HEA

Consider a 4-qubit HEA with linear connectivity (qubits 1-2-3-4) and 3 layers:

- Per layer: 4 qubits $\times$ 2 rotations/qubit = 8 single-qubit parameters.
- Per layer: 3 entanglers (CNOTs between 1-2, 2-3, 3-4).
- Total parameters: $3 \times 8 = 24$.
- Total CNOTs: $3 \times 3 = 9$.
- Circuit depth in two-qubit gates: 3 (one entangling layer per ansatz layer, applied in parallel where possible).

For H_2 at equilibrium geometry, this ansatz is expressive enough to reach within ~1 mHa of the FCI ground state, given a good initialization. For a random initialization, the optimizer typically gets stuck on a plateau and cannot reach chemical accuracy.

The same ansatz template, scaled to 8 qubits with the same layer count, has 48 parameters and 18 CNOTs. At 16 qubits: 96 parameters, 42 CNOTs. The parameter count grows linearly, but the gradient variance shrinks as $1/4^n$ — and somewhere between $n = 12$ and $n = 20$, depending on details, the optimizer simply cannot make progress from random initialization.

Visualization

Pending. A circuit diagram showing 4 horizontal qubit lines, with vertical "rotation columns" alternating with "CNOT staircases" — three layers total. Below: a plot of $\log \mathrm{Var}[\partial_i C]$ *vs* n *showing the* $O(1/4^n)$ *scaling. Caption: "Universal in the limit of large* L *— and that's the problem."*

Exercises

1. Count the parameters in an HEA with $n = 6$ qubits, ring connectivity, $L = 5$ layers, and 3 single-qubit rotations per qubit per layer. What is the CNOT count?
2. For a 2-qubit HEA $U(\theta) = \mathrm{CNOT} \cdot (R_Y(\theta_1) \otimes R_Y(\theta_2)) \cdot \mathrm{CNOT} \cdot (R_Y(\theta_3) \otimes R_Y(\theta_4))|00\rangle$, what is the dimension of the reachable manifold? Is the ansatz universal for 2-qubit pure states? Why or why not?
3. (VQA) The barren plateau theorem requires the ansatz to form an approximate 2-design. Give an intuition for why a random HEA with sufficient depth satisfies this condition, and explain why a problem-inspired ansatz (e.g., UCCSD) does not.

Research Extensions

- **Layer-wise initialization strategies** — train layer 1 to convergence before unfreezing layer 2, etc., to escape the plateau regime.
- **Symmetry-preserving HEAs** — modify the HEA to commute with conserved quantities, restricting to a smaller and more trainable manifold.
- **Hybrid HEA + warm start** — use a classical method (Hartree-Fock, MPS, neural network state) to initialize HEA parameters in a non-random region.

Cross-links to Other Courses

- **Random Matrix Theory** — Haar measure, t-designs, concentration of measure.
- **Statistics Module 4** — variance of estimators in high-dimensional spaces.
- **Quantum Hardware** — native gate sets, qubit connectivity graphs, coherence times.
- **Module 4 (Barren Plateaus)** — the formal pathology that this node introduces informally.

Chemically Inspired Ansatz

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.3

If the hardware-efficient ansatz (2.2) is "build whatever the device supports and hope the optimizer can deal," then the chemically-inspired ansatz is the opposite philosophy: **start from physics, write down the state-preparation recipe the chemistry suggests, and only then ask whether you can compile it.**

The canonical example is **Unitary Coupled Cluster with Singles and Doubles (UCCSD)**, the variational quantum analogue of the gold-standard classical post-Hartree-Fock method. A chemically-inspired ansatz pays for its inductive bias with depth — UCCSD circuits are much deeper than HEAs of comparable parameter count — but the bias is so well-chosen that for chemistry problems it can reach the answer where HEAs cannot.

This node generalizes from UCCSD to the broader class of **problem-inspired ansätze** and explains the structural moves that make them work.

The Coupled Cluster Idea

Classical coupled cluster (CC) theory expresses the ground state of a molecule as

$$|\Psi_{\mathrm{CC}}\rangle = e^{T}|\Phi_{\mathrm{HF}}\rangle,$$

where $|\Phi_{\mathrm{HF}}\rangle$ is the Hartree-Fock reference (a single Slater determinant) and T is a "cluster operator" that creates excitations:

$$T = T_1 + T_2 + T_3 + \ldots, \quad T_1 = \sum_{ia} t_i^a a_a^\dagger a_i, \quad T_2 = \sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i, \quad \ldots$$

Truncating after T_2 gives **CCSD** — couples singles and doubles excitations only. The amplitudes t_i^a, t_{ij}^{ab} are the variational parameters.

The classical CCSD method is **non-variational** — e^T is not unitary, and the truncation produces an energy estimate that can drop below the true ground state. The quantum analogue fixes this by replacing e^T with the **anti-Hermitian** version:

$$|\Psi_{\mathrm{UCCSD}}\rangle = e^{T-T^\dagger}|\Phi_{\mathrm{HF}}\rangle.$$

Now $T - T^\dagger$ is anti-Hermitian, so $e^{T-T^\dagger}$ is unitary, and the resulting state can be prepared by a quantum circuit. Crucially, the variational principle (1.1) now applies — the energy $\langle\Psi_{\mathrm{UCCSD}}|H|\Psi_{\mathrm{UCCSD}}\rangle$ is a true upper bound on the ground state.

How UCCSD Becomes a Circuit

To run UCCSD on a quantum device, you need to express $e^{T-T^\dagger}$ as a sequence of native gates. The standard recipe:

1. **Map fermionic operators to Pauli operators** via Jordan-Wigner or Bravyi-Kitaev (see node 2.5). Each excitation operator $a_a^\dagger a_i$ becomes a sum of Pauli strings.
2. **Trotterize the exponential.** Because the Pauli strings in $T - T^\dagger$ generally don't commute, you approximate $e^{T-T^\dagger}$ by a Trotter expansion: $\prod_k e^{i\theta_k P_k} + O(\Delta t^2)$.

3. **Implement each $e^{i\theta_k P_k}$ as a gate sequence** — this is a *Pauli rotation* and has a standard decomposition: a basis change to Z, a CNOT staircase, an $R_Z(\theta_k)$, then the inverse staircase and basis change.

For a small molecule like H_2 in a minimal basis (4 spin-orbitals, 2 electrons), the resulting UCCSD ansatz has:

- 2 single excitations and 1 double excitation (after applying spin and number symmetries),
- ~80 two-qubit gates after Jordan-Wigner mapping and Trotterization,
- 3 free parameters.

Compare this to a 4-qubit HEA with 3 layers: 24 parameters, 9 CNOTs. **UCCSD has many fewer parameters but a much deeper circuit.**

Why This Tradeoff Is Often Worth It

The UCCSD parameter manifold is *much smaller* than the full Hilbert space — it lives in the symmetry sector with the right particle number, spin, and (often) point group, and within that sector it covers exactly the states reachable by physically meaningful electron rearrangements.

Three concrete consequences:

1. No barren plateau in the same sense. The barren plateau theorem applies to ansätze that approximate Haar-random unitaries. UCCSD does not — it has heavy structural constraints — and the gradient variance is not exponentially small in qubit number for chemistry-typical instances. The optimizer can find descent directions even from the Hartree-Fock starting point.

2. Natural warm start. The Hartree-Fock state is the all-zero parameter point ($T = 0$ means $e^0 = I$), and Hartree-Fock is already a reasonable approximation to the ground state. The optimizer starts in a region where the gradient is informative and the energy is meaningful. HEAs have no such natural starting point.

3. Chemical interpretability. Each parameter corresponds to a specific physical excitation, so the converged amplitudes can be interpreted (large T_2 amplitudes correspond to strong electron correlation in specific orbital pairs, etc.). This is rare in quantum machine learning and rare in HEAs; it is a feature unique to problem-inspired ansätze.

What You Pay For It

The downsides of UCCSD-style ansätze, in order of increasing severity:

- **Circuit depth.** A Trotterized UCCSD circuit can be 10× deeper than a comparable HEA. On NISQ devices, depth means decoherence, and depth means more total error per evaluation.
- **Trotter error.** The Trotterization introduces $O(\Delta t^2)$ error in the prepared state. This error competes with the optimization error and bounds how close to the true ground state you can get without infinite Trotter steps.
- **Domain specificity.** UCCSD is designed for fermionic chemistry. For spin lattice models, lattice gauge theories, or QML problems, the right "problem-inspired" ansatz is something different (Hamiltonian Variational Ansatz, QAOA-style p-step ansätze, etc.). There is no universal recipe.
- **Compilation complexity.** Mapping $T - T^\dagger$ to Pauli strings and then to gates is non-trivial. Standard libraries (Qiskit Nature, OpenFermion, Pennylane) handle it, but the resulting circuit is fragile to manual modification.

Beyond UCCSD: ADAPT-VQE and Cousins

Even within chemistry, UCCSD is not the end of the story. The biggest practical extension is **ADAPT-VQE** (Grimsley et al. 2019):

> Start with the Hartree-Fock state. At each iteration, compute the gradient of the energy with respect to every operator in a pool (say, all singles + doubles). Add the operator with the largest gradient magnitude to the ansatz. Re-optimize all parameters. Repeat.

ADAPT-VQE *grows* the ansatz one operator at a time, choosing only the operators that actually help. The result is a much shorter circuit than UCCSD — for many molecules, ADAPT reaches chemical accuracy with 10-100× fewer gates. The price is that the structure of the ansatz is not known in advance; it emerges from the optimization itself.

ADAPT is a representative of a broader move: **adaptive ansatz construction**, where the parameterization is co-designed with the optimization rather than fixed at the outset. Module 10 returns to this when it discusses adaptive control views of VQAs.

Structural Role

This node is the second half of the "what kind of ansatz?" dichotomy. Together with 2.2, it spans the design space along the hardware-first-vs-problem-first axis. Subsequent nodes assume you understand both poles and can talk about a given ansatz in those terms.

It also forwards to node 2.5 (Hamiltonian encoding), since the construction of UCCSD-style ansätze depends on the same fermion-to-qubit mapping that the Hamiltonian uses.

Relations to Other Concepts

- **Coupled cluster theory** — the classical method UCCSD is the quantum analogue of.
- **Configuration interaction** — an alternative classical method based on a linear (rather than exponential) ansatz; less compact but variational.
- **Hamiltonian Variational Ansatz (HVA)** — for spin models, the analogue of UCCSD: layers of $e^{i\theta H_k}$ using the problem Hamiltonian itself as the generator.
- **QAOA** — the same idea as HVA applied to combinatorial optimization, with alternating problem and mixer layers.
- **Symmetry-adapted ansätze** — restrict to a particular symmetry sector (particle number, spin, parity) by construction.

Worked Example — H_2 UCCSD

In a minimal STO-3G basis, H_2 has 4 spin-orbitals (2 spatial × 2 spin). After Jordan-Wigner mapping, 4 qubits. Hartree-Fock state: $|1100\rangle$ (two electrons in the lowest spatial orbital, both spins). The relevant excitations:

- Single excitations: $a_2^\dagger a_0$, $a_3^\dagger a_1$ (2 amplitudes).
- Double excitation: $a_2^\dagger a_3^\dagger a_0 a_1$ (1 amplitude).

After symmetry reduction, the double excitation is the dominant degree of freedom — singles cancel for H_2 at equilibrium. A 1-parameter UCCD ansatz suffices for chemical accuracy:

$$|\psi(\theta)\rangle = e^{\theta(a_2^\dagger a_3^\dagger a_0 a_1 - a_0^\dagger a_1^\dagger a_2 a_3)}|1100\rangle.$$

After Jordan-Wigner, this becomes a sum of 8 Pauli strings, and after Trotterization, ~10 CNOTs and 1 R_Z. **1 parameter, ~10 gates, chemical accuracy on H_2.** A 4-qubit HEA needs ~24 parameters and at least 9 CNOTs to reach the same accuracy — and only with the right initialization.

The lesson: **inductive bias is worth a lot of parameters.**

Visualization

Pending. A circuit diagram of the H_2 UCCD ansatz: 4 qubits initialized to $|1100\rangle$, then a single Pauli rotation block. Annotation: "1 parameter, chemistry-aware, hits the answer." Side-by-side comparison with a 4-qubit HEA labeled "24 parameters, fights the optimizer."

Exercises

1. Show that for H_2 in a minimal basis, the singles excitations $a_2^\dagger a_0$ and $a_3^\dagger a_1$ do not contribute to the ground state at equilibrium geometry, by appealing to spin and orbital symmetry.
2. Why is the *anti-Hermitized* exponential $e^{T-T^\dagger}$ used in UCCSD rather than e^T? What property of the resulting state would be lost with the latter?
3. (VQA) ADAPT-VQE chooses operators greedily by gradient magnitude. Construct a toy example where this greedy choice is suboptimal — i.e., the operator with the largest gradient is not the one that should be added next.

Research Extensions

- **k-UpCCGSD** — a generalization of UCCSD that uses generalized singles and doubles (any spin-orbital pair) and allows multiple repetitions.
- **Hamiltonian Variational Ansatz (HVA)** — the spin-model analogue.
- **Operator pool design** — for ADAPT, the choice of pool fundamentally constrains what circuits can be built. Dynamic pools are an active research area.
- **Nonlinear UCC** — extending the ansatz with non-truncated cluster operators in restricted form.

Cross-links to Other Courses

- **Quantum Chemistry** — coupled cluster theory, electronic structure methods.
- **Group Theory / Symmetry** — particle number, spin, point group symmetries in fermionic systems.
- **Module 5 (Regularization)** — UCCSD-style ansätze pair particularly well with the thesis's regularization techniques because their landscapes are more benign.
- **Module 8 (Expressivity vs Trainability)** — UCCSD is an exemplar of "give up some expressivity for much better trainability."

Compilation, Depth, and Width

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.4

The mathematical ansatz $U(\boldsymbol{\theta})$ and the physical circuit that runs on a device are not the same thing. Between them sits a **compiler** — a piece of software that translates the abstract gate sequence into one the hardware can actually execute, respecting native gate set, qubit connectivity, scheduling constraints, and calibration.

This node is the bridge from "ansatz on paper" to "circuit on metal." The relevant concepts are **width** (how many qubits), **depth** (how many sequential gate layers), and **gate count** (how many physical operations). On NISQ devices, all three are budgets, and compiling well means staying inside all three at once.

Width

Width is the number of qubits the circuit acts on. On a real device, width is bounded by the number of available, well-calibrated qubits.

Three subtleties:

- **Logical vs physical qubits**: every algorithm-level qubit consumes one device qubit — no error-correction overhead in the NISQ setting. (Fault-tolerant compilation is a different story.)
- **Connectivity**: not all qubit pairs can interact directly. A "100-qubit device" might only support entangling gates between specific neighbors. If your ansatz wants a CNOT between two qubits that aren't connected, the compiler must insert SWAPs to bring them adjacent — and each SWAP is itself a CNOT-heavy operation.
- **Calibration heterogeneity**: not all qubits are equally good. A 27-qubit device might only have 12 qubits with low enough error rates to be worth using on a given day.

Width is the **easiest budget to violate** — your problem either fits or it doesn't — and the **hardest to compress** without physically larger hardware.

Depth

Depth is the length of the longest path through the circuit (the number of gate layers that must execute sequentially). On a real device, depth is bounded by the **coherence time** — the qubits remain in a usable quantum state only for a finite duration before decoherence destroys the information.

A typical NISQ device has:

- Single-qubit gate time: ~20 ns
- Two-qubit gate time: ~200-500 ns
- T1/T2 coherence: 50-200 µs

This gives a depth budget of **a few hundred two-qubit gates** before the state is irrecoverably noisy. For a 100-CNOT circuit, you have already consumed most of the coherence window.

Depth is the most-often-violated budget for chemistry-style ansätze, which is why UCCSD is hard to run on real hardware and ADAPT-VQE (and other depth-saving variants) exist.

Critical distinction: depth vs gate count. A circuit with 1000 single-qubit gates spread across 10 qubits has gate count 1000 but depth only ~100, because gates on different qubits run in parallel. Depth is the time-axis quantity; gate count is the resource quantity. Both matter, for different reasons.

Gate Count

Gate count is the total number of operations. It governs **noise accumulation** — each gate has some error probability p, so the circuit fidelity scales roughly as $(1-p)^{\text{gate count}} \approx e^{-p \cdot \text{gate count}}$.

For $p = 10^{-3}$ (typical NISQ two-qubit error) and a 1000-CNOT circuit, the fidelity is $e^{-1} \approx 0.37$ — meaning roughly 60% of your runs are corrupted by at least one error. For a 100-CNOT circuit, fidelity is $e^{-0.1} \approx 0.90$ — much more workable.

Gate count is what makes "shallow" ansätze attractive: not because they finish faster (a single circuit is fast either way), but because they accumulate less error per shot.

What the Compiler Does

A compiler takes an abstract $U(\boldsymbol{\theta})$ and produces a runnable circuit. Its responsibilities, in roughly the order they happen:

1. Decompose abstract gates into native gates. $R_Y(\theta)$ might not be native; only R_X, R_Z, and CZ might be. The compiler rewrites $R_Y(\theta)$ using the available basis: $R_Y(\theta) = R_Z(-\pi/2)R_X(\theta)R_Z(\pi/2)$.

2. Route logical qubits to physical qubits. The compiler picks an assignment of algorithm qubits to device qubits that minimizes SWAP overhead given the device connectivity graph.

3. Insert SWAPs where needed. For two-qubit gates between non-adjacent qubits, SWAPs move the qubits adjacent. Each SWAP = 3 CNOTs, so this is expensive.

4. Schedule. Decide which gates run in parallel (on different qubits) and which run sequentially. Minimizes depth.

5. Optimize. Combine adjacent rotations, cancel redundant CNOTs, exploit gate identities. Modern compilers use heuristic and SAT-based passes.

6. Calibrate-aware mapping. Pick the best available qubits given today's noise profile.

The compiled circuit can be **2-10× larger** than the abstract one for non-trivial connectivity, especially for chemistry-style ansätze that want all-to-all entanglement on a device with linear or grid connectivity.

The VQA Compilation Subtlety

In a normal quantum algorithm, you compile once, run many times. In a VQA, the *parameters* change between runs but the *structure* doesn't — so a smart VQA compiler compiles the **structure** once and only re-evaluates the parameter angles each iteration.

Modern toolchains (Qiskit, Pennylane, Cirq) all support this "parameterized compilation" pattern: build a circuit template with symbolic parameters, compile to native gates symbolically, then bind concrete parameter values per call. Without this, every optimizer step would pay full compilation cost — potentially seconds — and the inner loop would be unworkable.

This is one of those infrastructure details that doesn't show up in any paper but determines whether a 1000-iteration VQA finishes in an hour or a week.

Why This Matters For Optimizer Choice

Some optimizers ask for many evaluations per iteration (population methods, finite-difference gradient estimation): these are bottlenecked by per-evaluation cost. Others ask for few high-quality evaluations: these are bottlenecked by per-evaluation accuracy.

If your compiled circuit is large enough that every shot is expensive (e.g., it takes 100 ms of wall clock and consumes a chunk of your queue allocation), then **the right optimizer is one that uses few evaluations** — Bayesian optimization, surrogate methods, gradient methods with parameter-shift rather than finite differences.

If your circuit is small and cheap, you can afford a population method or a high-shot-budget gradient method.

The compiled circuit is the constraint that makes this tradeoff concrete. **You can't reason about VQA optimization in isolation from compilation cost** — it sets the per-iteration budget that all of the algorithmic choices have to fit inside.

Structural Role

This is the **operational reality check** node of Module 2. It connects the abstract ansatz definitions of 2.1-2.3 to the physical hardware that nodes 2.7-2.8 will describe in detail. Without this node, the rest of Module 2 reads as pure mathematics; with it, the design tradeoffs for the rest of the course have material weight.

It also feeds Module 9 (hardware noise models): noise rates and gate counts compose into the effective objective function noise profile that the optimizer faces.

Relations to Other Concepts

- **Quantum compilation** — the general subfield. Major tools: Qiskit transpiler, Pennylane transforms, Cirq optimizers, t|ket⟩.
- **Topological compilation** — fault-tolerant compilation for surface codes; not relevant in NISQ.
- **Gate teleportation / measurement-based computation** — alternative compilation models; not yet widespread.
- **Pulse-level compilation** — bypassing gate-level abstractions to compose at the analog pulse level; under active development.

Worked Example — UCCSD on a Linear Topology

Consider a 4-qubit H_2 UCCSD circuit. Abstract form: 1 Pauli rotation $e^{i\theta P}$ where P is an 8-Pauli-string sum acting on all 4 qubits.

After abstract decomposition: - 8 Pauli rotations (one per term in P, Trotterized to first order), - Each Pauli rotation = (basis change) + (CNOT staircase, length 3) + (R_Z) + (inverse staircase) + (inverse basis change), - ~7 CNOTs per term × 8 terms = ~**56 CNOTs**.

On a device with **all-to-all connectivity** (e.g., trapped ion), the compiled circuit is roughly that size. On a device with **linear connectivity** (qubits in a chain, 1-2-3-4), CNOTs between non-adjacent qubits need SWAPs: - A CNOT between qubits 1 and 4 costs 3 + 1 + 3 = 7 native CNOTs (SWAP across, do the gate, SWAP back). - If the staircase requires multiple non-adjacent CNOTs, the SWAP overhead compounds. - Compiled CNOT count: ~**120-180**, depending on routing.

So the same abstract UCCSD ansatz costs 2-3× more on a linear device than on a fully-connected one. This is the operational reason that trapped-ion devices are sometimes more competitive for chemistry than larger but less-connected superconducting devices, even at lower qubit count.

Visualization

Pending. Two-panel diagram. Left: an abstract circuit (4 qubits, a few gates, clean). Right: the same circuit after compilation onto a linear-topology device — many SWAPs inserted, longer overall depth. Caption: "Connectivity is a tax."

Exercises

1. Estimate the compiled depth and CNOT count for a 6-qubit HEA with 3 layers on a square 2×3 grid topology, assuming each layer's entangling pattern is a complete bipartite graph.
2. For a CNOT error rate of $p = 5 \times 10^{-3}$, what is the maximum CNOT count for which the circuit fidelity remains above 0.5?
3. (VQA) The "compile once, bind many" pattern is critical for VQA performance. Sketch a minimal API for parameterized compilation that supports gradient evaluation via the parameter-shift rule without recompilation.

Research Extensions

- **Approximate compilation** — accept some compilation error in exchange for shorter circuits; relevant for ansatz-level approximations.
- **Hardware-aware ansatz synthesis** — automatically construct an ansatz given a target Hamiltonian and a device connectivity graph.
- **Learned compilation** — RL-based compilers that improve on heuristic SWAP insertion.
- **Pulse-level VQAs** — bypass gate-level decomposition by directly optimizing pulse parameters.

Cross-links to Other Courses

- **Compilers (CS)** — instruction selection, register allocation, scheduling — the same algorithmic problems with quantum-specific cost models.
- **Module 9 (Hardware Noise Models)** — gate count maps to noise budget.
- **Quantum Hardware** — coherence times, native gate sets, connectivity graphs.
- **Operations Research** — SWAP minimization is a hard combinatorial problem.

Hamiltonian Encoding

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.5

The Hamiltonian H is the **definition of the problem**. But "the Hamiltonian" lives in different mathematical worlds depending on where you are in the pipeline:

- The chemist writes H in second-quantized form, using fermionic creation/annihilation operators.
- The condensed matter physicist writes H in spin variables, using Pauli operators directly.
- The combinatorial optimizer writes H as an Ising or QUBO cost.
- The VQA practitioner needs H as a **sum of Pauli strings on qubits**, because that is what the device can measure.

This node is about the **encoding step** — the translation from problem-native representation to a Pauli decomposition that a quantum device can act on. It is conceptually distinct from the ansatz (which prepares a state) and conceptually distinct from the optimizer (which acts on a scalar). The encoding is the third design choice that determines what the optimizer sees.

The Required Output

What every VQA needs is a representation of H of the form

$$H = \sum_{j=1}^{M} h_j P_j,$$

where each $h_j \in \mathbb{R}$ is a coefficient and each P_j is a tensor product of single-qubit Pauli operators (and identities) on n qubits — i.e., a "Pauli string."

This form is required because it is the only thing a quantum device can directly measure. Each Pauli string can be measured by rotating each qubit into the appropriate basis (Z for I/Z entries, H rotation for X entries, S†H rotation for Y entries) and reading out in the computational basis. The measured eigenvalue is the product of ± 1 outcomes per qubit.

Once H is in this form, the cost function

$$C(\boldsymbol{\theta}) = \langle \psi(\boldsymbol{\theta}) | H | \psi(\boldsymbol{\theta}) \rangle = \sum_j h_j \langle \psi(\boldsymbol{\theta}) | P_j | \psi(\boldsymbol{\theta}) \rangle$$

becomes a weighted sum of Pauli expectation values, each of which can be estimated by repeated measurement (see node 2.7).

The encoding step is therefore: **rewrite the problem Hamiltonian, in whatever form it started, as a Pauli sum.**

Encoding Spin Hamiltonians

For spin systems, the encoding is essentially trivial — the Hamiltonian is **already** a Pauli sum.

The transverse-field Ising model on n sites:

$$H = -J \sum_{\langle i,j \rangle} Z_i Z_j - h \sum_i X_i.$$

This is already in the required form. $M = |\text{edges}| + n$ terms, each a 1- or 2-local Pauli string.

The Heisenberg model:

$$H = J \sum_{\langle i,j \rangle} (X_i X_j + Y_i Y_j + Z_i Z_j).$$

Also already in the required form, with $M = 3|\text{edges}|$ terms.

For these problem classes, encoding is no work and the Pauli sum is short — $M = O(n)$. The qubit count equals the spin count.

Encoding Combinatorial Optimization

For QUBO problems (Quadratic Unconstrained Binary Optimization), the cost function has the form

$$f(\mathbf{x}) = \sum_i a_i x_i + \sum_{i<j} b_{ij} x_i x_j, \quad x_i \in \{0, 1\}.$$

To encode this as a quantum Hamiltonian, substitute $x_i = (1 - Z_i)/2$ so that $x_i = 0 \leftrightarrow Z_i = +1$ and $x_i = 1 \leftrightarrow Z_i = -1$. Expanding gives an Ising-like Hamiltonian:

$$H = c_0 + \sum_i c_i Z_i + \sum_{i<j} c_{ij} Z_i Z_j.$$

This is already a Pauli sum. $M = O(n^2)$ for dense problems, much less for sparse ones. The qubit count equals the number of binary variables. QAOA uses exactly this encoding.

Encoding Fermionic Hamiltonians

This is where the encoding step earns its name. A second-quantized fermionic Hamiltonian for an electronic structure problem looks like

$$H = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{pqrs} h_{pqrs} a_p^\dagger a_q^\dagger a_r a_s,$$

where $a_p^\dagger, a_q$ are fermionic creation/annihilation operators and the coefficients come from one- and two-electron integrals over molecular orbitals.

Fermions are not the same as qubits — the operators $a_p^\dagger, a_q$ **anticommute**, while qubit operators on different qubits **commute**. Translating between these algebras is the work of a **fermion-to-qubit mapping**.

There are three standard mappings.

Jordan-Wigner

Map each spin-orbital to one qubit. Define:

$$a_p^\dagger = \frac{1}{2}(X_p - iY_p) \otimes Z_{p-1} \otimes Z_{p-2} \otimes \cdots \otimes Z_0.$$

The trailing Z chain is what enforces fermionic anticommutation: each fermion operator carries a "parity string" through all earlier orbitals. This is the simplest mapping, conceptually: **one fermion per qubit**. Its weakness is that the Pauli strings are *long* — operators acting on widely separated orbitals produce Pauli strings spanning all qubits in between. For sparse fermionic Hamiltonians, JW gives long strings.

Bravyi-Kitaev

Use a more clever data structure (a binary tree of qubits) to encode parity information. Each fermion operator becomes a Pauli string of length $O(\log n)$ instead of $O(n)$. Trade-off: the implementation is more complex, and the resulting Pauli strings are *shorter on average* but spread across non-adjacent qubits, which can hurt on devices with local connectivity.

Parity Encoding

Encode the *partial sums of occupation numbers* rather than the occupations themselves. Useful in some contexts, less common in practice than JW or BK.

The choice between JW and BK matters in practice because it determines:

- Number of Pauli terms in the Hamiltonian (typically $O(n^4)$ for either, but with different prefactors).
- Locality of the Pauli strings (JW: linear; BK: $\log n$).
- Which strings can be measured simultaneously (see node 2.8 on measurement grouping).

Standard libraries (OpenFermion, Qiskit Nature) implement both and let you switch with a single function call.

Symmetry Reductions

After mapping to qubits, you can often **further reduce the qubit count** by exploiting symmetries.

For H_2 in a minimal STO-3G basis: - Naive Jordan-Wigner: 4 spin-orbitals $\to$ 4 qubits. - After exploiting parity (Z2 symmetries arising from spin and electron number conservation): **2 qubits**.

This is "qubit tapering" and is implemented in standard libraries. It reduces both the device cost and the parameter count of the ansatz, often by a factor of 2-4. For larger molecules the savings can be larger.

The cost is conceptual: the resulting 2-qubit Hamiltonian no longer has a clean physical interpretation as "electrons in orbitals." It's just a Pauli sum.

What The Optimizer Sees After Encoding

Once H is encoded, the optimizer sees a single scalar function $C(\boldsymbol{\theta})$ — *the encoding has disappeared.* The choice of mapping, the symmetry reductions, the qubit count: all of these are "compiled away" from the optimizer's perspective.

But the encoding choice has determined:

- **The number of Pauli terms**, which sets the per-evaluation shot cost (one shot budget per term, see node 2.8).
- **The variance structure** of the cost estimator, since variance depends on which Pauli terms are easy to measure jointly.
- **The qubit count**, which sets the depth budget for the ansatz.
- **The barren plateau profile**, indirectly, since fewer qubits means smaller Hilbert space and milder concentration.

So the encoding is invisible to the optimizer but it shapes everything the optimizer experiences. This is one of the places where Module 1's "stochastic objective generator" abstraction (1.8) is **leaky** — the noise structure of the generator is set by encoding decisions made elsewhere, and ignoring those decisions misses a major lever.

Structural Role

This node closes out the "what is the problem definition?" half of Module 2. Together with 2.1-2.3 (ansatz design) it answers: *what does the optimizer minimize, and over what?*

Forwards: - 2.7 (how measurement actually works) consumes Pauli strings as input. - 2.8 (measurement grouping and shot budget) consumes the *number* and *commutation structure* of the Pauli strings. - Module 9 (hardware noise models) consumes encoded Hamiltonians to study how noise propagates into objective error.

Relations to Other Concepts

- **Second quantization** — the standard formalism for many-body fermionic and bosonic systems.
- **Jordan-Wigner transformation** — historically the first fermion-to-qubit map, dating to 1928.
- **Bravyi-Kitaev transformation** — modern, more efficient mapping using parity trees.
- **Z2 symmetries / qubit tapering** — exploit conserved quantities to reduce qubit count.
- **Block encoding** — a different paradigm: encode H as a unitary block within a larger circuit. Used in fault-tolerant algorithms (qubitization), not VQAs.

Worked Example — H_2 End-to-End

Starting representation: second-quantized fermionic Hamiltonian for H_2 in STO-3G basis at 0.74 Å bond length.

After computing one- and two-electron integrals (PySCF or similar), the fermionic Hamiltonian has ~30 nonzero coefficients across ~8 distinct fermionic operators.

After Jordan-Wigner mapping to 4 qubits, this becomes a sum of **15 Pauli strings**:

$$H = -1.05 \cdot I - 0.39 \cdot Z_0 + 0.39 \cdot Z_1 - 0.01 \cdot Z_2 + \cdots + 0.18 \cdot X_0X_1Y_2Y_3 + \ldots$$

(approximate coefficients).

After Z2 symmetry tapering: **5 Pauli strings on 2 qubits**.

The optimizer doesn't know any of this. It sees a function from $\mathbb{R}^P$ to $\mathbb{R}$ and asks for evaluations. But each evaluation costs at least 5 measurement bases $\times$ N shots each. And the optimal ansatz on 2 qubits is much smaller than the optimal ansatz on 4 qubits. **The encoding choices set the cost basis and the design space.**

Visualization

Pending. A flow diagram. Top: "Fermionic Hamiltonian (8 distinct operators)". Middle: "Jordan-Wigner (15 Pauli strings, 4 qubits)". Bottom: "Symmetry-tapered (5 Pauli strings, 2 qubits)". Side annotations: "this affects ansatz design", "this affects measurement cost", "this determines what the optimizer sees".

Exercises

1. Apply the Jordan-Wigner transformation to the fermionic operator $a_2^\dagger a_0$ for a 4-orbital system. Write out the resulting Pauli string explicitly.
2. The Jordan-Wigner mapping gives $O(n)$-length Pauli strings; Bravyi-Kitaev gives $O(\log n)$. Estimate the asymptotic crossover qubit count where BK becomes preferable for measurement cost. (Hint: depends on the connectivity of the device and the dominant cost.)
3. (VQA) For a 2-qubit Pauli sum $H = c_0 II + c_1 ZI + c_2 IZ + c_3 ZZ + c_4 XX$, how many distinct measurement bases are needed to estimate $\langle H \rangle$? Which terms can be measured simultaneously?

Research Extensions

- **Custom fermion-to-qubit mappings** — tailoring the mapping to a specific device topology to minimize compiled circuit depth.
- **Locality-preserving mappings** — encoding Hamiltonians so that nearby orbitals map to nearby qubits.
- **Tapering automation** — automatic discovery of Z2 symmetries in arbitrary fermionic Hamiltonians.
- **Bosonic encodings** — encoding bosonic systems (vibrational modes, photonic networks) into qubits with bounded truncation error.

Cross-links to Other Courses

- **Quantum Chemistry** — second quantization, integrals, basis sets.
- **Group Theory** — Z2 symmetries, conserved quantities.
- **Module 8** — encoding choice affects expressivity.
- **Module 9** — encoding choice affects noise propagation.

Representations of Quantum States

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.6

When you simulate, debug, or analyze a VQA before running it on real hardware, you have to choose a **representation** for the quantum state. The same state has many possible mathematical descriptions, and each representation has a different cost-vs-fidelity tradeoff.

This node walks through the four representations that matter for VQA work:

1. **Statevector** — full amplitude vector
2. **Product state** — restricted to factorizable states
3. **Density matrix** — for mixed states and noise
4. **Matrix product state (MPS)** — for low-entanglement states

A working VQA practitioner uses all four at different points in the pipeline. Knowing when to use which is one of the most practically useful skills in the field, and it is rarely taught explicitly.

---

1. Statevector

The most direct representation: a complex vector

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \psi_x |x\rangle, \quad \psi_x \in \mathbb{C}, \quad \sum_x |\psi_x|^2 = 1.$$

For n qubits, this is a vector of 2^n complex numbers. Memory cost: $2^n \times 16$ bytes (double-precision complex) $= 16$ GB at $n = 30$, 16 TB at $n = 40$. Hard ceiling for personal computers around $n = 28 - 30$; for serious clusters, around $n = 40 - 45$.

Operations are matrix-vector multiplies. Applying a single-qubit gate is $O(2^n)$. Applying an n-qubit unitary directly is $O(4^n)$, though structured unitaries (a circuit of single- and two-qubit gates) can be applied gate-by-gate at $O(2^n)$ per gate.

Use for: debugging small instances, computing exact expectation values, sanity-checking optimizer behavior on toy problems where the ground truth is known.

Don't use for: anything past $n = 30$, or for simulating noise (no room for noise channels in a pure state).

The thesis's experimental setup uses 8-qubit statevector simulation for the VQE benchmarks. At 8 qubits, the statevector is 4 KB and operations are essentially free.

---

2. Product State (Tensor Product of Single-Qubit States)

Restricted to states of the form

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle.$$

Memory cost: $2n$ complex numbers (2 amplitudes per qubit). Linear in qubit count.

This representation is exact only for **non-entangled** states. For entangled states it is a strict approximation.

Use for: Hartree-Fock-style starting states (which *are* product states in the right basis), mean-field approximations, sanity-checking that an ansatz can break out of the product subspace.

Don't use for: any state past the first entangling layer of an ansatz. As soon as a CNOT runs, the state generally leaves the product subspace.

The product-state representation matters for VQAs mostly as a **starting point** or **classical baseline**, not as a working representation.

---

3. Density Matrix

For mixed states (statistical ensembles or open-system states), the right representation is

$$\rho = \sum_k p_k |\psi_k\rangle\langle\psi_k|, \quad p_k \geq 0, \quad \sum_k p_k = 1.$$

Memory cost: $2^n \times 2^n$ complex numbers $= 4^n \times 16$ bytes $= 16$ GB at $n = 15$, 16 TB at $n = 20$. Half the qubit ceiling of statevector, because the matrix has square as many entries.

Operations: unitary application is $\rho \to U\rho U^\dagger$, an $O(4^n)$ operation per gate. Noise channels are applied as Kraus sums, which is more general than statevector simulation can express.

Use for: noise simulation (mixed states are the natural language of decoherence), modeling realistic hardware behavior, computing fidelity to a target state.

Don't use for: problems past $n = 18$ or so, or when a pure-state representation suffices.

For the thesis's noise-aware simulations, density matrix representation is the workhorse.

---

4. Matrix Product State (MPS)

A clever approximation: write the amplitude tensor as a product of small matrices.

$$\psi_{x_1 x_2 \ldots x_n} = A^{(1)}_{x_1} A^{(2)}_{x_2} \cdots A^{(n)}_{x_n},$$

where each $A^{(i)}_{x_i}$ is a matrix of size $\chi_{i-1} \times \chi_i$ and χ_i is the **bond dimension** at site i.

For an exact representation of an arbitrary n-qubit state, you need bond dimension up to $2^{n/2}$ — exponential. But for **low-entanglement states**, a small bond dimension ($\chi = 16, 64, 256$) gives an excellent approximation.

The maximum half-chain entanglement entropy that an MPS with bond dimension χ can capture is $\log_2 \chi$. States obeying the **area law** for entanglement (most ground states of local Hamiltonians) can be efficiently represented this way; states with **volume-law entanglement** (random circuits past a certain depth, scrambled states) cannot.

Memory cost: $O(n\chi^2)$ — linear in qubit count, polynomial in bond dimension. Operations: $O(n\chi^3)$ per gate. Tractable for n up to many hundreds when χ is small.

Use for: simulating large 1D-like quantum systems, classical pre-optimization of ansätze, studying how entanglement grows with depth.

Don't use for: problems with strong long-range entanglement, deep random circuits, or states that genuinely live near the volume-law regime.

MPS simulation is what makes "100-qubit VQE" classically tractable for shallow ansätze on chemistry-style problems — and what makes "100-qubit barren plateau study" intractable.

How To Choose

A practical decision tree:

- **Are you debugging a small instance?** → Statevector.
- **Are you studying noise effects?** → Density matrix.
- **Is the state shallow / low-entanglement?** → MPS.
- **Are you computing a Hartree-Fock baseline?** → Product state.
- **Do you need exact answers and n is small enough?** → Statevector.
- **Is n large but the ansatz is shallow?** → MPS, then validate with statevector on a smaller instance.
- **Is the state highly entangled and n is large?** → You're in trouble. This is the regime where classical simulation fails and you actually need the quantum hardware.

The last point is the **quantum advantage frontier**: VQAs are interesting precisely when none of the classical representations can handle the state cheaply.

What The Hardware Returns

None of these representations is what the hardware itself stores. The hardware stores a physical quantum state, and you can only query it through measurement (node 2.7). You **never see** the statevector or density matrix from a real device — you see samples from a distribution induced by it.

This means:

- Debugging on a real device is much harder than debugging in simulation, because you can't inspect intermediate states.
- Comparing simulation to hardware requires statistically reconciling measurement outcomes with simulated probabilities.
- Process tomography (reconstructing the full state) is expensive ($O(4^n)$ measurements) and only practical for very small n.

In practice, VQA development happens in three stages:

1. **Statevector simulation** — verify the ansatz is correct, the optimizer works, the cost converges.
2. **Density matrix simulation** — add a noise model, see how noise affects convergence.
3. **Real hardware** — submit to actual device, deal with the gap between modeled and real noise.

Skipping any stage is asking for trouble.

Structural Role

This node is the **practitioner's toolbox** node of Module 2. It is mostly free of authored opinions and consists of standard background that any VQA worker needs. Forwards to 2.7 (measurement) which describes how the hardware exposes its state, and to 2.9 (ansatz as manifold) which uses the statevector representation to define the geometric picture.

It also feeds Module 4 (barren plateaus): the proofs of the barren plateau theorem use random unitary averages over the statevector representation, and being able to compute these averages depends on being able to write down the state.

Relations to Other Concepts

- **Tensor network states** — the broader family that MPS belongs to (PEPS for 2D, MERA for hierarchical, tree tensor networks for tree-like).

- **Stabilizer formalism** — a different polynomial classical representation, exact for Clifford circuits but unable to represent general states.
- **Neural network quantum states** — encode amplitudes as a neural network; can sometimes outperform MPS for non-1D structure.
- **Decoherence-free subspaces** — restricted state subspaces immune to particular noise types; useful for representation choice when noise structure is known.

Worked Example — Memory For 30 Qubits

How much memory does a 30-qubit state cost in each representation?

- **Statevector:** $2^{30} \times 16$ bytes = **16 GB**. Fits in a high-memory workstation.
- **Density matrix:** $2^{60} \times 16$ bytes = **16 EB**. Completely infeasible.
- **Product state:** 60 complex numbers = **480 bytes**. Trivial, but only correct for unentangled states.
- **MPS, bond dimension 64:** $30 \times 64^2 \times 16$ bytes $\approx$ **2 MB**. Trivial. But only correct if half-chain entanglement entropy is ≤ 6.
- **MPS, bond dimension 1024:** $30 \times 1024^2 \times 16$ bytes $\approx$ **500 MB**. Workable. Captures entanglement entropy up to 10.

Lesson: the order-of-magnitude difference between representations is larger than the order-of-magnitude difference between problem sizes. **Choosing the right representation is more important than buying more RAM.**

Visualization

Pending. A 4-quadrant plot. X-axis: qubit count. Y-axis: log memory cost. Four lines, one per representation, with characteristic crossovers and ceilings. Annotations on each line indicating "use this region."

Exercises

1. Show that a Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ cannot be written as a product state $|\psi_1\rangle \otimes |\psi_2\rangle$ for any choice of $|\psi_1\rangle, |\psi_2\rangle$.
2. What is the bond dimension required to represent the Bell state as an MPS? What about the GHZ state $\frac{1}{\sqrt{2}}(|00\ldots0\rangle + |11\ldots1\rangle)$ on n qubits?
3. (VQA) For a hardware-efficient ansatz with L layers on n qubits, give a heuristic argument for the minimum bond dimension needed to represent the resulting state to fixed accuracy as a function of L and n.

Research Extensions

- **Tree tensor networks for VQA** — alternative to MPS, sometimes better for branched problem structures.
- **Neural quantum states for VQA simulation** — using NQS as a classical baseline against quantum hardware.
- **Approximate density matrix methods** — low-rank density matrix representations for noisy state simulation at higher qubit count.

Cross-links to Other Courses

- **Quantum Information** — pure vs mixed states, density matrix formalism, partial traces.
- **Tensor Networks (separate course)** — MPS, PEPS, MERA, contractions.
- **Numerical Methods** — large-vector arithmetic, sparse matrix tricks.
- **Statistics Module 7** — high-dimensional state representations and their information geometry.

How Measurement Actually Works in a VQA

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.7

This node is for the reader who has seen $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ written down a hundred times and is finally ready to ask: **what physically happens, on a real device, in order to produce that scalar?**

The mechanics are not glamorous. They involve a lot of basis changes, a lot of computational-basis readouts, a lot of bit-counting, and a lot of statistical aggregation. But understanding them concretely is what separates a practitioner from a paper-reader, and it is the *why* behind several key facts that show up later in the course (variance scaling, measurement grouping, shot noise, hardware bias).

The Goal

We want to estimate

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle, \quad H = \sum_{j=1}^{M} h_j P_j.$$

The strategy: estimate each Pauli expectation $\langle P_j\rangle$ separately, then take the linear combination. This works because expectation is linear: $\langle H\rangle = \sum_j h_j \langle P_j\rangle$.

So the entire problem reduces to: **how do you measure a Pauli string P on a state prepared by a circuit?**

Measuring a Single Pauli String: The Recipe

For a Pauli string like $P = X_0 \otimes Z_1 \otimes Y_2 \otimes I_3$ on 4 qubits, the recipe is:

Step 1: Prepare $|\psi\rangle$. Run the ansatz circuit $U(\boldsymbol{\theta})$ on the device, starting from $|0\rangle^{\otimes n}$.

Step 2: Rotate each non-Z, non-I qubit into the Z basis. - For each X entry in P: apply a Hadamard gate H to that qubit. - For each Y entry in P: apply $S^\dagger H$ to that qubit (equivalently $R_X(\pi/2)$). - For each Z entry: do nothing — it's already in the Z basis. - For each I entry: do nothing — measurement of identity is trivial (always $+1$).

After this rotation, the Pauli string has been mapped to a string of Z's and I's, and measuring it in the **computational basis** (which is the Z basis on every qubit) gives the eigenvalue of P.

Step 3: Read out every qubit in the computational basis. The hardware projects each qubit onto $|0\rangle$ or $|1\rangle$, giving a bitstring $b \in \{0,1\}^n$.

Step 4: Compute the eigenvalue from the bitstring. For each qubit position where P has a non-identity entry (X, Y, or Z after rotation, all now Z), the eigenvalue contribution is $(-1)^{b_i}$ ($+1$ for outcome 0, -1 for outcome 1). The full Pauli eigenvalue is the product over all non-identity positions:

$$\text{eigenvalue}(b) = \prod_{i \in \text{support}(P)} (-1)^{b_i}.$$

This is ± 1.

Step 5: Repeat N times and average. Run the circuit N times — each is a "shot" — and average the eigenvalues:

$$\widehat{\langle P \rangle} = \frac{1}{N} \sum_{k=1}^{N} \text{eigenvalue}(b^{(k)}).$$

This is your estimator. It is unbiased: $\mathbb{E}[\widehat{\langle P \rangle}] = \langle P \rangle$. Its variance is bounded by $1/N$ (since each sample is in $[-1, 1]$).

What This Looks Like For The Whole Hamiltonian

Multiplying the protocol across all M Pauli strings:

```
For each Pauli string P_j in H:
    For shot = 1 to N_j:
        Prepare |psi(theta)>
        Rotate to P_j's measurement basis
        Read out the bitstring
        Compute the eigenvalue
    Average to get hat<P_j>

Final cost estimate = sum_j h_j * hat<P_j>
```

Per cost evaluation: $M \times N$ total circuit runs, where M is the number of Pauli strings and N is the per-string shot count. For H_2 (5 strings after tapering), with $N = 1000$ shots each, that's 5000 circuit executions per cost evaluation.

Each evaluation takes wall-clock time on the order of seconds to minutes on a real device, depending on queue depth and circuit length.

Variance and the $1/\sqrt{N}$ Scaling

The estimator $\widehat{\langle P \rangle}$ is a sample mean of ± 1 random variables. By standard statistics:

$$\text{Var}[\widehat{\langle P \rangle}] = \frac{1 - \langle P \rangle^2}{N}, \quad \text{StdDev}[\widehat{\langle P \rangle}] = \sqrt{\frac{1 - \langle P \rangle^2}{N}} \leq \frac{1}{\sqrt{N}}.$$

The standard deviation shrinks as $1/\sqrt{N}$. **This is the fundamental scaling that bounds VQA cost evaluation accuracy.** To halve the error, quadruple the shots.

For the full Hamiltonian, the variance is

$$\text{Var}[\widehat{\langle H \rangle}] = \sum_j \frac{h_j^2 (1 - \langle P_j \rangle^2)}{N_j} \leq \sum_j \frac{h_j^2}{N_j}.$$

If you spend the same N shots on each term, the total cost-estimator variance is bounded by $\frac{1}{N} \sum_j h_j^2$. For molecules, $\sum_j h_j^2$ can be on the order of 1-100 hartree^2, which means even moderate accuracy (millihartree) requires hundreds of thousands of shots per evaluation.

This is a very tight constraint, and it is the reason **measurement grouping** (node 2.8) is so important — it is the only way to amortize measurements across multiple Pauli terms and bring the per-evaluation shot cost into the workable range.

What Hardware Adds On Top

The recipe above assumes ideal hardware. On a real device, several extra effects show up:

1. Gate errors during state preparation. Each gate in $U(\boldsymbol{\theta})$ has some error rate p_g. After 100 gates with $p_g = 10^{-3}$, the prepared state has fidelity $\approx e^{-0.1} \approx 0.9$ to the ideal state. The measured expectation values are biased toward those of the noisy state.

2. Readout error. The "measure in the Z basis" step itself has an error rate, typically 1-5%. A qubit that is actually in $|0\rangle$ is reported as 1 with probability p_{readout}, and vice versa. This adds bias to all expectation values.

3. Crosstalk. Measuring one qubit can subtly disturb the others, creating correlations between supposedly independent measurements.

4. Calibration drift. Gate parameters and readout fidelities change over time. A measurement made at the start of a queue may have slightly different bias than one made an hour later.

All of these turn the ideal recipe into a noisy approximation. The key consequence: **the estimator is biased, not just noisy.** The bias depends on the hardware and changes over time. Unbiased estimation via post-processing (readout error mitigation, zero-noise extrapolation, probabilistic error cancellation) is an active research area.

What The Optimizer Sees

After all of this, the optimizer is handed a single number per evaluation: $\widehat{C}(\boldsymbol{\theta}_k)$. This number is

- biased (by hardware noise and possibly by mitigation procedures),
- noisy (by shot statistics and time-correlated drift),
- delayed (by queue and execution time),
- expensive (each evaluation consumed $M \times N$ circuit runs).

This is the operational meaning of the "stochastic objective generator" abstraction from node 1.8. The abstraction is correct, but **the numbers behind it come from a long pipeline that you have to design to control variance and bias**.

The thesis's experimental setup on simulated hardware fixes the shot count at $N = 1024$ per term and runs for tens of thousands of total circuit executions per optimization run. On real hardware, the equivalent can take hours to a day.

Structural Role

This is the **operational unmasking** node of Module 2. It exposes what $\widehat{C}(\boldsymbol{\theta})$ actually means in terms of physical operations, and connects the abstract Pauli decomposition (2.5) to the noisy scalar the optimizer sees.

Forwards to 2.8 (measurement grouping), which is the optimization step that comes immediately after this raw recipe.

Relations to Other Concepts

- **Quantum measurement (Born rule)** — the underlying physics. $P(b|\psi) = |\langle b|\psi\rangle|^2$.
- **POVMs (positive operator-valued measures)** — the most general measurement formalism; the protocol above is a special case.
- **Direct fidelity estimation** — protocols for estimating state fidelity using random Pauli measurements.

- **Classical shadows** — efficient protocols that estimate many observables from a single measurement set; an active alternative to per-Pauli measurement.
- **Readout error mitigation** — post-processing to remove bias from readout errors.

Worked Example — Measuring $\langle X_0 Z_1 \rangle$ on a Bell State

Suppose $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and we want to estimate $\langle X_0 Z_1 \rangle$.

Step 1: prepare the Bell state via $H_0, \mathrm{CNOT}_{0,1}$ on $|00\rangle$.

Step 2: rotate qubit 0 (X position) into Z basis by applying H. Qubit 1 (Z position) needs no rotation.

The state after rotation is:

$$H_0|\psi\rangle = H_0 \cdot \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|+0\rangle + |-1\rangle) = \frac{1}{2}(|00\rangle + |10\rangle + |01\rangle - |11\rangle).$$

Step 3: measure both qubits in computational basis. Each outcome (b_0, b_1) occurs with probability $|\text{amplitude}|^2 = 1/4$.

Step 4: the eigenvalue is $(-1)^{b_0}(-1)^{b_1}$. - $00 \to +1$ (prob 1/4) - $10 \to -1$ (prob 1/4) - $01 \to -1$ (prob 1/4) - $11 \to +1$ (prob 1/4)

Expectation: $\frac{1}{4}(+1 - 1 - 1 + 1) = 0$.

So $\langle X_0 Z_1 \rangle = 0$ for the Bell state. (Sanity check: this is correct — the Bell state has zero expectation for any Pauli string except the four stabilizers $II, ZZ, XX, -YY$.)

Step 5: repeat N times. The sample mean converges to 0 with variance $1/N$.

Visualization

Pending. A circuit diagram with three layers labeled: "(1) state prep — $U(\boldsymbol{\theta})$", "(2) basis change — H, S†H per Pauli letter", "(3) Z-basis readout". A bracket below indicating "this entire circuit runs N times". A separate annotation on the right showing the bitstring → eigenvalue computation.

Exercises

1. For the state $|\psi\rangle = R_Y(\theta)|0\rangle$, describe the measurement protocol for estimating $\langle X \rangle$ and $\langle Z \rangle$. Compute the variance of each estimator at $\theta = \pi/2$ for $N = 100$ shots.
2. Show that $\mathrm{Var}[\widehat{\langle P \rangle}] = (1 - \langle P \rangle^2)/N$. Where in the derivation does the boundedness of the eigenvalues to ± 1 enter?
3. (VQA) The naive cost-estimator variance bound is $\sum_j h_j^2/N$. For a 4-qubit H_2 Hamiltonian with 15 Pauli terms, estimate the shot count per term needed to achieve millihartree accuracy in the cost estimator.

Research Extensions

- **Classical shadows** — measure random unitaries, estimate many observables at once; logarithmic shot scaling in some regimes.
- **Adaptive shot allocation** — dynamically reallocate shots between Pauli terms based on observed variance.

- **Importance-weighted sampling** — query terms with larger $|h_j|$ more often.
- **Bayesian estimators** — use prior knowledge about $\langle P_j \rangle$ to improve early-iteration accuracy.

Cross-links to Other Courses

- **Quantum Measurement** — Born rule, projective measurements, POVMs.
- **Statistics Module 3** — sample mean, variance of estimators, central limit theorem.
- **Module 8** (measurement grouping) — the immediate next step after this node.
- **Module 9** (hardware noise models) — what gets layered on top in practice.

Measurement Grouping and Shot Budget

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.8

The naive measurement protocol from node 2.7 — measure each Pauli term in the Hamiltonian separately, with N shots each — is operationally feasible but **wasteful**. For a Hamiltonian with M Pauli terms, it costs $M \times N$ circuit runs per cost evaluation. For chemistry-style Hamiltonians where M can be in the hundreds or thousands, this is prohibitive.

The remedy is **measurement grouping**: identify Pauli terms that can be measured *simultaneously* in the same circuit run, group them, and amortize the shot cost across the group. Doing this well can reduce the per-evaluation cost by an order of magnitude.

This node explains the grouping problem, the standard solutions, and how a finite shot budget gets allocated across groups in practice.

When Two Pauli Strings Can Be Measured Simultaneously

Two Pauli strings can be **measured in the same experiment** if and only if they commute and there exists a single basis change that diagonalizes both. The simplest sufficient condition is **qubit-wise commutativity (QWC)**: two Pauli strings are QWC if at every qubit position, either:

- they have the same Pauli letter (XX, YY, ZZ, II), or
- at least one of them is the identity (XI, IY, ZI, II, etc.).

If two strings are QWC, then the per-qubit basis change for one is compatible with the per-qubit basis change for the other, and a single measurement in the combined basis gives you the bitstring needed to compute eigenvalues for **both**.

Example: $P_1 = X_0 Z_1$ and $P_2 = X_0 I_1$. These are QWC — both have X on qubit 0, and P_2 has identity on qubit 1. A single measurement (rotate qubit 0 to Z basis via H, leave qubit 1 alone, read out both) gives bitstrings from which you compute both $\widehat{\langle P_1 \rangle}$ (using both bits) and $\widehat{\langle P_2 \rangle}$ (using only bit 0).

QWC is the **simplest** notion of joint measurability. There are stronger notions:

- **General commutativity**: two Pauli strings commute as operators, but the diagonalizing basis change is not local (it requires entangling gates in the basis-change layer).
- **Anticommuting partitions**: with extra ancilla qubits, you can measure even mutually anticommuting sets of Paulis jointly.

The thesis's regularization techniques don't depend on grouping strategy specifically, but the shot budget allocations they assume are based on standard QWC grouping.

The Grouping Problem

Given a Hamiltonian $H = \sum_j h_j P_j$, partition the Pauli strings into the **minimum number of QWC groups**. Each group can be measured by a single circuit (with a single basis-change layer); within a group, every shot contributes to estimating every term in the group.

This is exactly the **graph coloring problem** on a "compatibility graph": vertices are Pauli strings, edges connect non-QWC pairs. A valid grouping is a coloring; the minimum grouping is the chromatic number.

Graph coloring is NP-hard in general, but for chemistry Hamiltonians:

- A greedy coloring runs in $O(M^2)$ and produces near-optimal groupings in practice.

- Specialized algorithms (Sorted Insertion, Largest-First Degree-Sorted) consistently reduce M from "thousands" to "tens" for typical chemistry Hamiltonians.

For H_2 (15 Pauli terms after Jordan-Wigner): grouping reduces this to **2 groups**, meaning a 7.5× reduction in circuit count per evaluation. For larger molecules like BeH_2: groupings of ~10-20 from raw counts of 200-500. For very large Hamiltonians (FeMoco, biomolecules): hundreds of groups from tens of thousands of terms.

The reduction is not free — within a group, terms with different supports compete for the same shots, and the variance allocation has to be done carefully — but it is a major operational win.

Variance Within a Group

Suppose a group G contains Pauli terms $P_{j_1}, P_{j_2}, \ldots, P_{j_K}$. A single measurement in the group's basis produces a bitstring from which you compute one eigenvalue per term:

$$\widehat{e}_k = \prod_{i \in \text{support}(P_{j_k})} (-1)^{b_i}, \quad k = 1, \ldots, K.$$

After N_G shots:

$$\widehat{\langle P_{j_k} \rangle} = \frac{1}{N_G} \sum_{s=1}^{N_G} \widehat{e}_k^{(s)}.$$

The crucial point: **all K estimators come from the same shots**. Their variances are still each bounded by $1/N_G$, but they are **correlated** with each other (because they share the underlying measurement outcomes).

The total variance of the cost contribution from the group is:

$$\text{Var}\left[\sum_{k=1}^{K} h_{j_k} \widehat{\langle P_{j_k} \rangle}\right] = \frac{1}{N_G} \text{Var}_{\text{shot}}\left[\sum_k h_{j_k} \widehat{e}_k\right],$$

where the inner variance is over the joint distribution of one shot's worth of measurements in the group basis.

For QWC groups, the inner variance can be substantially smaller than $\sum_k h_{j_k}^2$ (the bound for independent measurements) when the Pauli terms are correlated under the group basis. So measurement grouping does not just save circuit runs — it can also **reduce the per-shot variance contribution**.

Allocating a Shot Budget

Given a total shot budget N_{total} per cost evaluation and a set of G groups, how should you allocate shots across groups?

Two reasonable strategies:

Equal allocation. Give each group $N_G = N_{\text{total}}/G$. Simple, and works fine when the per-group variance contributions are roughly equal.

Variance-weighted allocation. Give each group a number of shots proportional to its variance contribution:

$$N_G = N_{\text{total}} \cdot \frac{\sigma_G}{\sum_{G'} \sigma_{G'}},$$

where σ_G is an estimate of the per-shot standard deviation of the group's contribution. This minimizes the total variance for a fixed total shot count and is the optimal allocation in the sense of Lagrange multipliers. It is sometimes called the "Wu et al." or "Rubin" allocation.

In practice, optimal variance-weighted allocation requires knowing variances in advance, which you don't. A common compromise: use a few hundred shots per group as a "scout" to estimate variances, then allocate the remaining budget proportionally.

The Shot Budget As An Optimization Hyperparameter

The total shot budget N_{total} per evaluation is **part of the optimization design**, not a fixed number imposed by hardware. Tradeoffs:

- **More shots** → less estimator noise → cleaner gradient signal → faster per-iteration progress.
- **Fewer shots** → cheaper evaluations → more iterations per wall-clock budget → more total parameter updates.

Most VQA optimizer choices make implicit assumptions about this tradeoff:

- **SPSA** (Simultaneous Perturbation Stochastic Approximation) tolerates very low shot counts and many iterations.
- **COBYLA** assumes moderate shot counts and uses geometry-based updates.
- **Adam with parameter-shift gradients** wants high enough shot counts that the gradient is reliable.
- **HOPSO** (the thesis's preferred optimizer) tolerates noisy evaluations and benefits from more iterations.

The thesis runs experiments at fixed $N_{\text{total}} = 1024$ shots per evaluation to enable apples-to-apples comparison across optimizers. Real-hardware deployments often use **shot scheduling** — start with low shots (cheap, noisy) for early exploration and ramp up as the optimizer converges (expensive, accurate near the minimum). See node 5.7 for the thesis-level treatment.

What Goes Wrong If You Get This Wrong

Two common failure modes:

1. Insufficient shots per term. With $N = 100$ shots and a Pauli term whose variance is near 1, the estimator standard deviation is ~0.1. For an optimizer trying to detect a gradient component of magnitude 0.05, this is not enough — the noise drowns the signal, and the optimizer wanders. Symptom: cost trajectory looks like a random walk rather than a descent.

2. Imbalanced shot allocation. If you give the 14-Pauli-term H_2 Hamiltonian 100 shots per term but the dominant term has $|h_j| = 10$ and the rest have $|h_j| < 0.1$, then the dominant term contributes $100\times$ more uncertainty to the cost estimate than the rest combined — you have wasted 90% of your shot budget on terms whose noise contribution is negligible. Symptom: variance-weighted reallocation produces a 5-10$\times$ efficiency gain.

Both failure modes are extremely common in beginner VQA work and very rarely diagnosed correctly. Watching the per-term variance contribution to the cost estimate is the single most informative debugging signal.

Structural Role

This is the **operational closure** node of Module 2. Module 2 has now described, end to end, what happens when the optimizer requests a cost evaluation: ansatz design (2.1-2.3) → compilation (2.4) → encoding (2.5) → state representation (2.6) → measurement protocol (2.7) → shot budget and grouping (this node).

Forwards: Module 5 (regularization) uses the shot budget as the resource against which improvements are measured. Module 9 (hardware noise) revisits the variance contributions when realistic noise models are layered in.

Relations to Other Concepts

- **Graph coloring** — the formal problem behind QWC grouping.
- **Importance sampling** — variance-weighted shot allocation is its discrete analogue.
- **Classical shadows** — an alternative to grouping that estimates many observables from random measurements.
- **Adaptive measurement protocols** — schemes that change basis based on previous outcomes.

Worked Example — Grouping for H_2

After Jordan-Wigner, the H_2 Hamiltonian (4 qubits) has 15 Pauli terms.

Let's enumerate a few: - $I_0I_1I_2I_3$ (identity, contributes a constant) - Z_0, Z_1, Z_2, Z_3 (single-qubit Z's) - Z_0Z_1, Z_0Z_2, Z_0Z_3, Z_1Z_2, Z_1Z_3, Z_2Z_3 (ZZ pairs) - $X_0X_1Y_2Y_3$, $X_0Y_1Y_2X_3$, $Y_0X_1X_2Y_3$, $Y_0Y_1X_2X_3$ (XXYY-type four-body)

The 11 Z-only terms are all mutually QWC (they all only have Z's and I's on every qubit) — they form **one group**, measured by reading out all 4 qubits in the Z basis.

The 4 XXYY terms are also mutually QWC: each has the same Pauli letter at each position (after checking carefully — they aren't quite, actually). In practice, they form 1-2 additional groups depending on the exact strings.

Total grouping: **2-3 groups**. From 15 separate measurements down to 2-3 — a roughly 5-7× speedup.

For a 100-shot budget per group (a generous allocation), the total cost per evaluation is 200-300 circuit runs instead of 1500. **That difference is the difference between a workable VQA inner loop and an unworkable one.**

Visualization

Pending. A graph diagram. Vertices: 15 Pauli strings labeled. Edges: connect non-QWC pairs. The vertices are colored to show a 2-coloring (the QWC grouping). Below: a bar chart showing "without grouping: 15 measurements" vs "with grouping: 2 measurements".

Exercises

1. For the Pauli strings $\{X_0Z_1, X_0Y_1, Z_0Z_1, X_0X_1\}$, determine which pairs are QWC. Find the minimum QWC grouping.
2. Show that for a Hamiltonian made entirely of Z-string terms (all Pauli operators are Z or I), every term is QWC with every other term, so the grouping reduces to a single group.
3. (VQA) The greedy QWC grouping algorithm has worst-case approximation ratio bounded by the maximum degree of the conflict graph. Sketch why this matters for chemistry Hamiltonians vs spin-model Hamiltonians.

Research Extensions

- **General-commutativity grouping** — beyond QWC, allowing entangling basis-change layers.
- **Anticommuting set measurements** — using ancilla qubits to measure mutually anticommuting Paulis.
- **Fragment-based methods** — partition the Hamiltonian into structurally meaningful fragments (one-body, two-body, etc.) and measure each fragment optimally.
- **Online shot allocation** — reallocate shots adaptively as the optimization proceeds.

Cross-links to Other Courses

- **Graph Theory** — coloring, chromatic number.
- **Operations Research** — resource allocation under uncertainty.
- **Statistics Module 9** — experimental design, optimal allocation of measurements.
- **Module 5 (Regularization)** — uses the shot budget as the headline cost metric.

The Ansatz as a Manifold

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Parameterized Circuits **Node:** 2.9

This is the **closing node** of Module 2 and its most geometric. Everything in the module so far has been described in coordinates: parameter vectors, Pauli strings, gate sequences, bitstrings, shot counts. This node steps back from coordinates and looks at the underlying geometric object: **the set of states the ansatz can prepare is a smooth submanifold of complex projective space, and many of the hardest optimization phenomena in VQA are properties of that manifold rather than properties of any particular parameterization.**

This is the geometric viewpoint that makes Module 3 (cost landscapes), Module 4 (barren plateaus), and Module 6 (optimization dynamics) feel coherent rather than ad-hoc. If you internalize this node, the rest of the course is a series of statements about a single geometric object that you can mentally visualize.

The Manifold

For an ansatz $U(\boldsymbol{\theta}) : \mathbb{R}^P \to U(2^n)$ acting on $|0\rangle^{\otimes n}$, the **ansatz manifold** is the image:

$$\mathcal{M} = \{|\psi(\boldsymbol{\theta})\rangle : \boldsymbol{\theta} \in \mathbb{R}^P\} \subset \mathbb{CP}^{2^n-1}.$$

We work in complex projective space $\mathbb{CP}^{2^n-1}$ rather than the unit sphere of $\mathbb{C}^{2^n}$ because global phases don't matter physically — $|\psi\rangle$ and $e^{i\alpha}|\psi\rangle$ are the same state.

Three structural facts about this set:

1. Dimension. The manifold has intrinsic dimension at most P, the parameter count. It can have lower dimension if some parameters are redundant (the parameterization is *singular* somewhere) or if the ansatz contains an exact symmetry that closes off a direction.

2. Smoothness. For circuit ansätze built from analytic gate families (rotations, controlled rotations), the parameterization $\boldsymbol{\theta} \to |\psi(\boldsymbol{\theta})\rangle$ is real-analytic almost everywhere. The manifold is smooth except at isolated singular points.

3. Restriction. $\mathcal{M}$ is a strict, generally low-dimensional subset of $\mathbb{CP}^{2^n-1}$, which itself has dimension $2(2^n - 1)$ — exponential in qubit count. For $n = 10$ qubits, the ambient projective space has real dimension 2046, while a typical 30-parameter ansatz manifold has dimension 30 (or less). **The ansatz traces out a thin 30-dimensional sheet inside a 2046-dimensional space.** Optimization is the act of moving along this sheet looking for a low-energy point.

Tangent Vectors and the Quantum Fisher Information

At a point $|\psi(\boldsymbol{\theta})\rangle$ on the manifold, the tangent space is spanned by the directional derivatives

$$|\partial_i \psi\rangle := \frac{\partial}{\partial \theta_i} |\psi(\boldsymbol{\theta})\rangle.$$

These vectors live in the ambient $\mathbb{C}^{2^n}$ but project (after removing the component parallel to $|\psi\rangle$) into the tangent space of $\mathcal{M}$ at $|\psi(\boldsymbol{\theta})\rangle$.

The natural metric on the manifold is the **Fubini-Study metric**, which on the tangent space takes the form

$$g_{ij}(\boldsymbol{\theta}) = \mathrm{Re}\langle \partial_i \psi | \partial_j \psi \rangle - \langle \partial_i \psi | \psi \rangle \langle \psi | \partial_j \psi \rangle.$$

In quantum information this is known as the **quantum Fisher information matrix** (up to a factor of 4) and it measures how "distinguishable" two infinitesimally separated parameter values are in terms of physical quantum measurements.

The crucial point: this metric is **not** the Euclidean metric δ_{ij} on parameter space. Different parameter directions span tangent vectors of very different lengths in the ambient state space. A "step of size ϵ in parameter space" can correspond to a tiny physical change in the state (if the parameter is in a redundant or weakly-coupled direction) or a huge physical change (if the parameter directly steers the state).

This mismatch is the reason **vanilla gradient descent in parameter space is not optimal** for VQAs. The natural object to descend is the cost gradient pulled back through the inverse metric:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\, g^{-1}(\boldsymbol{\theta}_k)\, \nabla C(\boldsymbol{\theta}_k).$$

This is the **quantum natural gradient** (Stokes et al. 2020) and it is treated in detail in Module 6. For now, the important takeaway is: **the natural geometry of the ansatz lives on the manifold, not in parameter coordinates.**

Singularities

A point $\boldsymbol{\theta}_0$ is **singular** if the tangent vectors $\{|\partial_i \psi(\boldsymbol{\theta}_0)\rangle\}_{i=1}^{P}$ do not span a P-dimensional subspace — that is, if some directional derivative collapses to zero, or two directions become parallel.

At a singular point: - The local dimension of the manifold drops below P. - The metric g_{ij} becomes singular (determinant zero). - The natural gradient is undefined (the inverse metric blows up). - Multiple distinct $\boldsymbol{\theta}$-paths can lead to the same $|\psi\rangle$, creating a degeneracy.

These singularities are not abstract pathologies — they show up in real ansätze. Typical examples: - **Identity points**: a layer with all rotation angles set to 0 contributes the identity matrix and removes itself from the parameterization. - **Symmetry-equivalent parameters**: two parameters whose effects on the state are related by an exact group action. - **Compositional cancellations**: $R_X(\theta)R_X(-\theta) = I$ creates a 1-parameter family of $(\theta, -\theta)$ configurations all mapping to the same state.

The barren plateau phenomenon (Module 4) is *not* the same as a singularity — it is a global statistical phenomenon — but it can be exacerbated by the manifold passing through near-singular configurations.

Why This Matters For Optimization

Three operationally important consequences of the manifold view:

1. Step size in parameter space $\neq$ step size in state space. A learning rate η that is appropriate in one region of parameter space may be too aggressive or too conservative in another, depending on local manifold geometry. This is why adaptive learning rates (Adam, AdaGrad) often help VQAs — they implicitly normalize for changing local geometry.

2. Curvature drives convergence rate. Just as in classical optimization, the local curvature of the manifold (and the cost surface on it) determines how quickly an optimizer can converge. Highly curved regions need small steps; flat regions allow large steps. Vanilla gradient descent ignores this; second-order methods (natural gradient, Newton-Raphson, BFGS) try to exploit it.

3. The manifold is the same regardless of parameterization. Two ansätze that produce the same set of reachable states have the same manifold — even if their parameterizations are very different. **The**

geometry is intrinsic; the coordinates are convention. This is the fundamental reason that the *choice of parameterization* matters even when it doesn't change what states can be reached: it changes the relationship between parameter directions and tangent directions, and therefore changes which optimizers will be efficient.

The Picture for HEAs and UCCSDs

Two quick intuitive snapshots:

HEA manifold. Roughly Haar-uniform on the symmetry sector reachable by the ansatz. As depth grows, the manifold fills out more and more of the unit sphere in Hilbert space. At the depth where it approximates a 2-design, the manifold is "everywhere dense" in a useful sense — and that's exactly when barren plateaus appear. The manifold is too uniform; almost no direction is special.

UCCSD manifold. Heavily structured by the Hartree-Fock reference and the chosen excitations. Sits in a low-dimensional symmetry sector (fixed particle number, spin). Has natural starting point (the HF state, where all parameters are zero) and natural directions of motion (each excitation amplitude). The manifold is small but well-shaped, and the cost function has identifiable minima rather than uniform plateaus.

The contrast: **HEA gives you a featureless manifold; UCCSD gives you a featured one.** The featured manifold is harder to derive but easier to optimize.

Structural Role

This is the **conceptual capstone** of Module 2. It pulls Module 2's coordinate-level facts (gates, parameters, depths, encodings, shots) up into a coordinate-free geometric picture.

Forwards: - Module 3 (Cost Landscapes) treats the cost function $C : \mathcal{M} \to \mathbb{R}$ as a function on the manifold and studies its critical points and topology. - Module 4 (Barren Plateaus) is a statement about the typical curvature of the cost function on the manifold under random parameterization. - Module 6 (Optimization Dynamics) asks how to update parameters in a way that respects manifold geometry rather than parameter coordinates. - Module 8 (Expressivity vs Trainability) is the formal version of "featureless vs featured manifold."

This is the node where Module 2 stops being about plumbing and starts being about geometry.

Relations to Other Concepts

- **Information geometry** — the general theory of metric structures on parameterized probability/quantum models. Amari and Nagaoka are the standard references.
- **Riemannian manifolds** — the mathematical setting for the natural gradient and second-order methods.
- **Complex projective space** $\mathbb{CP}^{2^n-1}$ — the ambient space in which the ansatz manifold lives.
- **Fubini-Study metric** — the canonical metric on $\mathbb{CP}^n$, and the source of the quantum Fisher information.
- **Stiefel manifold** — alternative parameterization for orthonormal frames, occasionally used in tensor-network analogues of ansatz design.

Worked Example — Two Manifolds With The Same Reach

Consider two single-qubit ansätze:

Ansatz A: $|\psi_A(\theta,\phi)\rangle = R_Z(\phi)R_Y(\theta)|0\rangle$. Two parameters. Reaches every pure single-qubit state. The manifold is the **full Bloch sphere**.

Ansatz B: $|\psi_B(\alpha,\beta,\gamma)\rangle = R_Y(\gamma)R_Z(\beta)R_Y(\alpha)|0\rangle$. Three parameters. Also reaches every pure single-qubit state. The manifold is **also the full Bloch sphere**.

Same manifold. Different parameterizations.

What differs: - **Local metric.** At a generic point, the metric g_{ij} for ansatz A is full rank 2×2. For ansatz B it is rank-2 in a 3×3 matrix — there is one redundant direction at every point. - **Optimization behavior.** A gradient-descent run on ansatz A walks across the Bloch sphere in 2D. The same algorithm on ansatz B has an extra "null direction" (the redundant one) that the optimizer can drift along without changing the state. - **Parameter-space distance.** Two states close together on the Bloch sphere can correspond to parameter vectors that are far apart in either ansatz — but the relationship is different in the two parameterizations.

Both manifolds are the same set of physical states. **The cost landscape differences come entirely from how the parameters cover the manifold.** This is one of the cleanest demonstrations that the choice of parameterization matters even when the reachable set does not.

Visualization

Pending. A diagram showing the Bloch sphere with two coordinate grids overlaid: one from ansatz A (latitude/longitude-like, smooth), one from ansatz B (a 3D parameter space being projected, with a "redundant fiber" at each point shown as a small circle). Caption: "Same manifold, different coordinates, different gradient flows."

Exercises

1. For the 2-qubit ansatz $U(\theta_1,\theta_2) = (R_Y(\theta_1) \otimes R_Y(\theta_2))|00\rangle$, compute the tangent vectors $|\partial_1\psi\rangle$ and $|\partial_2\psi\rangle$. Are they orthogonal? What is the dimension of the resulting manifold?
2. Show that the parameterization $R_X(\theta_1)R_X(\theta_2)|0\rangle$ has a 1-dimensional kernel everywhere — i.e., the linear combination $|\partial_1\psi\rangle - |\partial_2\psi\rangle = 0$. What does this imply about the dimension of the manifold?
3. (VQA) Sketch an argument for why the natural gradient $g^{-1}\nabla C$ is invariant under reparameterizations of the ansatz, while the Euclidean gradient ∇C is not.

Research Extensions

- **Quantum information geometry of variational models** — formalizing the manifold view across ansatz families.
- **Geodesic optimization on ansatz manifolds** — using exact or approximate Riemannian optimizer steps that follow geodesics rather than straight lines in parameter space.
- **Manifold curvature and barren plateaus** — connecting global geometric properties to optimization difficulty.
- **Riemannian regularization** — adding curvature-aware penalties that act intrinsically rather than in parameter coordinates.

Cross-links to Other Courses

- **Differential Geometry** — manifolds, tangent spaces, metrics, curvature.
- **Information Geometry** — Fisher metric, dual connections, exponential families.

- **Module 3 (Cost Landscapes)** — the cost function as a scalar field on the manifold.
- **Module 6 (Optimization Dynamics)** — the natural gradient as the manifold-aware update rule.

Module 3 — Cost Landscapes

Cost Function Design

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.1

This is the **opening node** of Module 3 and the place where the question shifts from *what is the ansatz?* to *what is the function we are minimizing on it?*

The cost function in a VQA is **not** uniquely determined by the problem. You start with a problem (find the ground state of H) and you have some freedom to choose what scalar you actually optimize against. This node walks through that freedom, the standard choices, and the consequences each choice has for the resulting landscape.

It is a node about **design decisions you have to make before the optimizer ever sees a number**.

The Default: Energy Expectation

The textbook VQA cost function is the energy expectation:

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle.$$

Why this is the default: - It is bounded below by the true ground state E_0 (variational principle). - It has a meaningful physical interpretation as the expected energy of the prepared state. - It is linear in the Pauli decomposition of H, so it can be computed by Pauli-by-Pauli measurement (node 2.7). - It is the cost the variational principle directly addresses.

For 95% of VQA work — VQE for chemistry, VQE for spin models, ground-state preparation — this is what you optimize.

When The Default Isn't Enough

There are several situations where the energy expectation is a *bad* cost function and you have to design something else.

Excited states

The variational principle gives you the *ground* state. To find an excited state, you need a cost function whose minimum is the excited state of interest. Three approaches:

Subspace-search VQE. Optimize against $\mathrm{Tr}(H\rho_S)$ over states in a fixed subspace, with constraints enforcing orthogonality to lower states.

Variational Quantum Deflation. Add penalty terms to the energy expectation that push the state away from already-found lower states:

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle + \beta \sum_{i\in\text{found}} |\langle\psi_i|\psi(\boldsymbol{\theta})\rangle|^2.$$

The penalty term raises the cost of any state with overlap with previously found eigenstates.

SSVQE. Train multiple states simultaneously with constraints on their orthogonality.

Each of these is a cost function design choice that changes the landscape topology.

Dynamics simulation

For preparing a state evolved by e^{-iHt} variationally, the cost is not energy but **fidelity to a target state**:

$$C(\boldsymbol{\theta}) = 1 - |\langle \psi_{\text{target}} | \psi(\boldsymbol{\theta}) \rangle|^2.$$

Or, computed in another way, the **infidelity in time-evolved measurements**.

These cost functions have qualitatively different landscape structures from the energy expectation — they are bounded above as well as below, they have explicit fixed points at $\boldsymbol{\theta}_0 = \arg\min C$, and they typically have *more* local minima than the energy expectation.

Quantum machine learning

For QML problems (variational classifiers, generative models), the cost function is a problem-specific loss:

- **Cross-entropy** for classification.
- **MSE** for regression.
- **MMD or Wasserstein** for generative models.

The cost is a function of the *measured outputs* of the circuit, which themselves are expectation values, but the relationship between parameters and cost is now mediated by a classical loss function. This adds another layer of nonlinearity (and another source of design freedom).

Constrained optimization

When the problem has symmetry constraints (particle number, total spin), you can enforce them either:

- **Implicitly**, by restricting the ansatz to states obeying the symmetry, OR
- **Explicitly**, by adding penalty terms to the cost: $C(\boldsymbol{\theta}) = \langle H \rangle + \mu \langle (N - N_0)^2 \rangle$.

The penalty approach changes the landscape — it creates new minima at the constrained points and lifts the energy of states outside the constraint surface.

Cost Function Design as an Optimization Move

The key insight: **the cost function is itself a hyperparameter.**

Two cost functions that nominally encode the same problem can produce dramatically different optimization behaviors: - different barren plateau profiles, - different numbers of local minima, - different gradient magnitudes, - different convergence rates.

Most VQA papers fix the cost function early and then spend the rest of their effort optimizing the optimizer. A better question is sometimes: **could we have chosen a different cost function whose landscape is friendlier?**

Examples of cost-function-side improvements:

- **Local cost functions** (Cerezo et al. 2021): replace a global cost like $\langle H \rangle$ with a sum of local 2-qubit observables. Provably avoids barren plateaus for shallow ansätze.
- **Variance penalization** (node 5.7 in this course): add a term proportional to the *variance* of the cost estimator, which keeps the optimizer in regions where the cost is more reliably measurable.
- **Cost function re-normalization**: shift and rescale the cost so its dynamic range matches the optimizer's expectations. Surprisingly effective for adaptive learning rate methods.

Each of these is a *cost-side* intervention — modify the function being optimized rather than the algorithm doing the optimizing. It is the dual to optimizer-side interventions and often the more impactful one when it works.

Cost Function Design and Module 5

This connects directly to Module 5's regularization thesis. The thesis distinguishes between two kinds of regularization:

- **Prior injection** — modify the cost function (objective-side intervention).
- **Dynamical modification** — modify the optimizer's update rule (dynamics-side intervention).

The cost function design choices in this node are all examples of prior injection. They change C before the optimizer sees it. Whether this is a good idea — and how it interacts with the optimizer's dynamics — is exactly what Module 5's thesis is about.

So this node is the **upstream** of Module 5's main argument: it is the place where the practitioner consciously chooses what objective they are optimizing, and that choice determines what kind of regularization makes sense.

Structural Role

This is the opening node of Module 3 and the bridge from the **physical** content of Module 2 (how do we get a number out of the device?) to the **landscape** content of Module 3 (what does the resulting function look like?).

Forwards to 3.2 (topology), 3.3 (minima), and to Module 5 (regularization, where cost-function-side modification appears in full).

Relations to Other Concepts

- **Loss function design in ML** — same problem, different domain. Cross-entropy vs MSE vs hinge vs focal loss is the classical analogue.
- **Penalty methods in classical optimization** — Lagrangian relaxation, augmented Lagrangians, barrier functions.
- **Variational principle (1.1)** — guarantees the energy expectation is a defensible default.
- **Surrogate cost functions** — replace the true cost with a cheaper-to-evaluate proxy.

Worked Example — Two Costs For The Same Problem

For H_2, consider two cost functions:

Cost A: Energy expectation.

$$C_A(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H_{H_2}|\psi(\boldsymbol{\theta})\rangle.$$

Range: [-1.85, +0.5] hartree. Minimum at $\boldsymbol{\theta}^*$: -1.137 hartree (FCI). Barren plateau profile (HEA): yes, in the random-init regime.

Cost B: Energy expectation + variance penalty.

$$C_B(\boldsymbol{\theta}) = \langle H\rangle + \lambda \mathrm{Var}[\widehat{\langle H\rangle}].$$

For the same minimizer, the variance is small (eigenstates have zero variance for their corresponding eigenvalue), so the penalty pushes toward eigenstate-like configurations. Range: same as above, plus a positive correction. Minimum: still near the FCI ground state, but the basin is *narrower* and *deeper* — variance vanishes only at eigenstates.

Which is better? Cost A converges faster initially (gradient is larger). Cost B converges slower but gets less stuck in mid-energy plateaus. The right answer depends on the optimizer being used and the problem regime.

This is the kind of tradeoff that cost function design can navigate — and that the rest of Module 3 will give you the tools to reason about.

Visualization

Pending. Two side-by-side cost surfaces. Left: pure energy expectation, with mild gradient and a wide basin. Right: energy + variance penalty, with steeper gradient and narrower basin. Caption: "The cost function is a design choice, not a given."

Exercises

1. For a 1-parameter ansatz $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$ and Hamiltonian $H = Z$, compute both $\langle H\rangle(\theta)$ and $\mathrm{Var}[\langle H\rangle](\theta)$. Where are the minima of each?
2. The variational principle gives $\langle H\rangle \geq E_0$. Does the same kind of guarantee hold for the variance-penalized cost? Argue carefully.
3. (VQA) Design a cost function whose minimum is the *first excited state* of H, given knowledge of the ground state $|\psi_0\rangle$. What new local minima does your construction introduce?

Research Extensions

- **Adaptive cost reformulation** — reshape the cost during optimization based on observed difficulty.
- **Cost function bandits** — treat the choice of cost function as an arm to be explored.
- **Dual cost functions** — optimize a relaxation, then refine on the original.
- **Cost-shaping for QML** — designing classification losses tuned to circuit ansatz structure.

Cross-links to Other Courses

- **Machine Learning** — loss function design as a first-class concern.
- **Optimization Theory** — penalty methods, Lagrangian duality.
- **Module 5 (Regularization)** — the natural continuation of this node.
- **Module 8 (Expressivity vs Trainability)** — local vs global cost functions and barren plateaus.

Landscape Topology

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.2

Once you have a cost function (3.1), you can ask: **what does it look like as a function on parameter space?** The answer is a **landscape** — a real-valued function over a (typically high-dimensional) torus, with a topology determined jointly by the ansatz, the Hamiltonian, and the cost choice.

This node introduces the basic topological vocabulary that the rest of Module 3 will use, and characterizes the qualitative features that VQA landscapes exhibit.

It is a vocabulary node, not a theorem node. The point is to give you the words for what you're seeing when you look at a cost surface — *basin*, *ridge*, *saddle*, *plateau*, *period* — and to anchor the intuition that these features are not random but are determined by the structure of the underlying variational problem.

Domain: A Torus, Not a Vector Space

The first topological fact about VQA cost functions: their domain is a **torus**, not Euclidean space.

For an ansatz built from rotation gates $R_P(\theta)$ where each Pauli P has eigenvalues ± 1, the cost function satisfies

$$C(\boldsymbol{\theta} + 2\pi \mathbf{e}_i) = C(\boldsymbol{\theta})$$

for every parameter θ_i. This periodicity makes the natural domain $\mathbb{T}^P = (S^1)^P = [-\pi, \pi]^P$ with edges identified.

Practical consequences:

- **No "boundary" of parameter space.** Every direction wraps around. The optimizer cannot "go to infinity."
- **Parameter-norm regularization is awkward.** L2 penalties $\lambda\|\boldsymbol{\theta}\|^2$ prefer one specific origin, but on a torus there is no privileged origin — every point is equivalent topologically.
- **Multiple paths between any two points.** On a torus, you can go around the long way or the short way, and an optimizer with momentum can sometimes exploit this.
- **Topology contributes minima.** A function on a torus has, by Morse theory, a minimum number of critical points determined by the topology — at minimum, one minimum, one maximum, and (for $P \geq 2$) several saddle points. **You cannot have a torus with only one critical point.**

The torus is a standard mathematical setting for periodic optimization but is rarely emphasized in VQA papers. It quietly determines a lot of what is and isn't possible.

Five Qualitative Features

Every VQA landscape exhibits some mixture of the following features.

1. Basins

A **basin** is a region around a local minimum where the gradient points (statistically) toward the minimum. Optimizers that find a basin will descend to its bottom and stay there unless they can escape.

VQA landscapes typically have **many basins**, ranging from very shallow (small region of attraction, mild depth) to deep (large region, low energy). The structure of basins is the most important thing for an optimizer's behavior, more important than the global minimum location.

2. Ridges and Saddles

A **saddle point** is a critical point where the Hessian has both positive and negative eigenvalues. The cost decreases in some directions and increases in others. Gradient descent slows down at saddles but eventually escapes them through the descent directions; SGD can escape faster due to noise.

A **ridge** is a region of high cost separating two basins. Crossing a ridge requires either going around it (energy-conserving path) or briefly increasing the cost (energy-violating path). Most local optimizers cannot cross ridges; this is what makes them get stuck in suboptimal basins.

For typical VQA ansätze, the ratio of saddle points to local minima can be quite high — much of the "structure" of a high-dimensional landscape is saddle-shaped, and avoiding saddles is a real concern.

3. Plateaus

A **plateau** is a region where the gradient is small in all directions. Plateaus come in two flavors:

- **Topological plateaus** are regions of genuinely small gradient — flat regions of the function. They occur near critical points and in "valleys" between minima.
- **Concentration plateaus** (or barren plateaus, Module 4) are regions where the gradient is exponentially small *in expectation* — a statistical property of how the cost function is constructed, not a feature of any particular point.

Both look the same to a local optimizer: the gradient is too small to drive movement. Distinguishing them requires global information that the optimizer doesn't have.

4. Periodic Repeats

Because the domain is a torus, every feature appears infinitely many times (one per fundamental cell). A local minimum at $\boldsymbol{\theta}^*$ is also at $\boldsymbol{\theta}^* + 2\pi\mathbf{e}_i$ for every i. This means an optimizer with periodic awareness can move between fundamental cells "for free" — sometimes useful for escaping a plateau by jumping a half-period.

In practice, most optimizers are unaware of the periodic structure and treat parameter space as Euclidean. This is a small operational loss.

5. Symmetry-Equivalent Critical Points

If the ansatz has a symmetry — e.g., parameters θ_i and θ_j play interchangeable roles for some pair (i, j) — then critical points come in equivalence classes under that symmetry. A local minimum at (θ_1^*, θ_2^*) is also a local minimum at (θ_2^*, θ_1^*). This is treated in detail in node 3.4.

What VQA Landscapes Look Like in Aggregate

Combining all of the above, the typical VQA landscape has the following statistical character:

- **Many local minima.** An ansatz with P parameters typically has $O(P)$ or more local minima for chemistry-style cost functions.
- **Many more saddles.** Saddle points outnumber minima by a substantial factor in high dimensions.
- **Mostly flat in expectation.** For random initializations of expressive ansätze, the *average* gradient magnitude is small compared to the global cost range.
- **Non-trivial basin sizes.** The basins of attraction of local minima vary in size from "the global minimum with a basin filling most of parameter space" to "tiny isolated minima with negligible attraction."
- **Symmetry-related structure.** Many critical points are equivalent under exchange or sign symmetries.

This is qualitatively similar to neural network loss landscapes (as studied in deep learning theory), and many of the heuristics from that field transfer. But the *detailed* structure is determined by the ansatz design and the Hamiltonian, and it cannot be assumed from the analogy alone.

Why This Matters For Optimizer Choice

The features above interact differently with different optimizers:

- **Gradient descent** does well in smooth basins, fails on plateaus and saddles, cannot cross ridges.
- **Adam / momentum methods** can build up enough velocity to escape mild saddles and climb small ridges.
- **Population-based methods** (HOPSO, evolutionary strategies) maintain diverse points across multiple basins simultaneously.
- **Bayesian optimization** is good at navigating landscapes with few minima but expensive evaluations; struggles with high-dimensional plateau regions.
- **SPSA** is robust to noise and can escape some plateaus through random perturbation.

The thesis's argument that the choice of optimizer is the dominant variable in many VQA experiments (node 1.9) is in part a statement about how different optimizers handle the topology features above. **Two optimizers running on the same landscape can have qualitatively different trajectories because they engage with the topology differently.**

Structural Role

This node provides the **vocabulary** for the rest of Module 3. The next nodes elaborate specific features:

- 3.3: local vs global minima in detail.
- 3.4: how symmetries shape the landscape.
- 3.5: gradients and curvature, the local structure.
- 3.6: concentration of measure, the statistical structure.
- 3.7: shot noise, the stochastic structure.
- 3.8: parameter-shift rule, the gradient computation.

You'll come back to this node's vocabulary repeatedly. It is the lens.

Relations to Other Concepts

- **Morse theory** — the topology of smooth functions on manifolds. Counts critical points by index and relates them to the manifold's homology.
- **Loss landscape topology in deep learning** — analogous study of the loss landscapes of neural networks. Exhibits many of the same features.
- **Random landscape theory** — physical models of random Gaussian functions on high-dimensional spaces. Predicts saddle/minimum ratios in high dimension.
- **Spin glass landscapes** — the canonical model of a "rugged" landscape with many critical points.

Worked Example — A 2D VQE Landscape

For $H = Z_0 + Z_1 + 0.5X_0X_1$ on 2 qubits with ansatz $|\psi(\theta_1, \theta_2)\rangle = R_Y(\theta_1) \otimes R_Y(\theta_2)|00\rangle$:

$$C(\theta_1, \theta_2) = \cos\theta_1 + \cos\theta_2 + 0.5 \sin\theta_1 \sin\theta_2.$$

This landscape on the 2-torus $[-\pi, \pi]^2$ has:

- A **global minimum** near $(\theta_1, \theta_2) = (\pi - \arctan(0.25), \pi - \arctan(0.25))$ where both Z's are -1 and the XX correlation aligns.
- A **second local minimum** related by the symmetry $(\theta_1, \theta_2) \to (-\theta_1, -\theta_2)$.
- **Two saddle points** near $(0, \pi)$ and $(\pi, 0)$ where the Z's disagree.
- **A maximum** near $(0, 0)$ where everything is $+1$.
- **Periodic copies** of all of the above shifted by 2π in each direction.

You can visualize this surface as a 2D heatmap and see all of these features clearly. **This is the structure your optimizer is navigating, even when you can't visualize it.**

Visualization

Pending. A 2D heatmap of the worked example, showing minima (dark), maxima (light), saddles (mid), with periodic boundaries. Annotated with the topological vocabulary words: basin, ridge, saddle, plateau.

Exercises

1. For the 2D landscape $C(\theta_1, \theta_2) = \cos\theta_1 + \cos\theta_2$, enumerate all critical points on the fundamental cell $[-\pi, \pi]^2$ and classify them as min/max/saddle.
2. Show that on a 1-torus S^1, any continuous function must have at least one minimum and one maximum (this is a special case of Morse inequalities). Generalize to the 2-torus.
3. (VQA) Sketch how a basin's *size* differs from its *depth*, and explain why an optimizer might prefer a wide-shallow basin over a narrow-deep one even when the latter has lower energy.

Research Extensions

- **Topological data analysis of cost landscapes** — using persistent homology to characterize landscape structure across ansatz families.
- **Critical point counting for variational ansätze** — analytic results for specific ansatz classes.
- **Connectivity of basins** — when are two basins connected by descent paths in the cost-restricted manifold?
- **Mode connectivity** — borrowing ideas from neural network landscapes about connected sets of low-cost points.

Cross-links to Other Courses

- **Topology / Morse Theory** — critical points, indices, homology.
- **Statistical Physics** — random landscapes, spin glasses, replica calculations.
- **Machine Learning** — loss landscape geometry of deep networks.
- **Module 4 (Barren Plateaus)** — concentration plateaus in detail.

Local and Global Minima

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.3

This node zooms in on the most important topological feature of a VQA landscape — its **minima** — and dissects what they mean for optimization.

The mantra "we want to find the global minimum" is correct in the abstract but operationally misleading in VQA practice. For most realistic VQA instances, the global minimum is **unreachable** by local optimization, and the practical question is **which local minimum can the optimizer find, and how good is it?**

Definitions

A **critical point** $\boldsymbol{\theta}^*$ of $C : \mathbb{T}^P \to \mathbb{R}$ satisfies $\nabla C(\boldsymbol{\theta}^*) = 0$.

A critical point is a **local minimum** if there exists a neighborhood U of $\boldsymbol{\theta}^*$ such that $C(\boldsymbol{\theta}^*) \leq C(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in U$.

A critical point is a **strict local minimum** if the inequality is strict for $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$.

A **global minimum** is the lowest local minimum on the entire domain.

The **basin of attraction** of a local minimum $\boldsymbol{\theta}^*$ is the set of starting points $\boldsymbol{\theta}_0$ from which gradient descent (with appropriately small step size) converges to $\boldsymbol{\theta}^*$.

How Many Minima?

A typical VQA cost function on P parameters has **many** local minima — usually more than P, often growing polynomially or even exponentially in P.

This is true for two related reasons:

1. Symmetry. Ansatz symmetries create equivalence classes of minima. If the ansatz has a permutation symmetry over k parameters, there are at least $k!$ symmetry-equivalent minima for any non-symmetric configuration. Permutation-symmetric HEAs at moderate qubit count can have hundreds of equivalent minima.

2. Compositional cancellation. Layers in a deep ansatz can compose to (approximately) the identity in many distinct ways, creating local minima at parameter configurations whose effective state is similar. $R_X(\theta_1)R_X(\theta_2)$ has a 1-parameter family of (θ_1, θ_2) configurations that all give the same single-qubit unitary, and any cost function on this combined parameterization inherits a 1-parameter degeneracy.

The number of minima grows roughly with the number of "ways" the ansatz can prepare each state, which for expressive ansätze is large.

How Good Are The Local Minima?

A more important question than "how many?" is **how good are they?**

For UCCSD-style chemically-motivated ansätze, the result is often striking: **almost all local minima are within chemical accuracy of the global minimum.** That is, any optimizer that finds *any* local minimum is essentially "done." This is one of the key practical advantages of problem-inspired ansätze.

For HEA-style hardware-motivated ansätze, the result is much worse: local minima can range from "barely better than random" to "near the global minimum," with the typical value far above the global minimum and the gap growing with depth. HEAs do have many local minima, but most of them are mediocre.

This is the **quality of local optima** problem, and it is one of the main reasons HEAs are usually outperformed by problem-inspired ansätze on chemistry instances even when both ansätze are equally trainable. The HEA *can* find a low-cost point, but the typical low-cost point it lands in is much higher than the typical low-cost point of UCCSD.

Why You (Usually) Can't Find The Global Minimum

For high-dimensional non-convex problems, finding the global minimum is generally NP-hard. For VQA problems specifically, several additional obstacles apply:

1. Exponentially many local minima. Even if you could count them, you can't afford to visit them all.

2. Saddles between basins. Local optimizers cannot cross saddles upward. To get from one basin to another, you need either restart-based exploration or a global method.

3. Noisy evaluations. You don't actually know whether a point is a local minimum or just a region with small gradient. Distinguishing requires more shots.

4. Limited query budget. Even global methods need many queries to explore a high-dimensional space, and each query is expensive.

In practice, VQA practitioners do **multi-start optimization**: run gradient descent (or another local method) from many random initializations, take the best result. This is not guaranteed to find the global minimum but typically finds a good one. Algorithms like HOPSO and CMA-ES are essentially principled multi-start strategies that share information between restarts.

The "Reachability vs Findability" Distinction

A critical conceptual move: **a state being reachable on the manifold is not the same as a state being findable by the optimizer.**

For an ansatz that contains the true ground state in its image: - **Reachable**: there exists $\boldsymbol{\theta}^*$ such that $|\psi(\boldsymbol{\theta}^*)\rangle$ equals the ground state. - **Findable**: starting from a typical initialization, the optimizer converges to $\boldsymbol{\theta}^*$ (or to a parameter giving a state with energy near the ground state) within the available query budget.

The first is a property of the ansatz manifold (Module 2.9). The second is a property of the *combination* of the ansatz, the cost function, the optimizer, and the initialization.

A barren plateau is the extreme case of "reachable but not findable" — the manifold contains the answer, but the cost gradient is so small everywhere that no local optimizer can detect a descent direction.

This is also exactly the conceptual move that node 1.9 (separation of objective and dynamics) demands: reachability is an objective property, findability is a composition property.

What An Optimizer Actually Settles On

In practice, when a VQA optimizer "converges," it has found one of the following:

1. **A local minimum** of the noisy cost function (which may or may not be a local minimum of the noiseless cost).

2. **A noise-equilibrium point** where the gradient signal is below the noise floor and the optimizer's updates cancel out on average.
3. **A point in a long flat region** where the gradient is small but not zero, and the optimizer is moving slowly.

Distinguishing these is hard. A common heuristic is to look at the variance of the cost over the last K iterations: if it is dominated by the optimizer's step size, you are at a noise equilibrium; if it is dominated by shot noise, you are at a stable point.

Practical Implications

Three operational rules of thumb that follow from the above:

1. Always do multi-start. Single-start optimization wastes the structure of the landscape — you can pay the cost of multiple optimizations in parallel and at least know which basin you've found.

2. Don't trust convergence as quality. A "converged" run might be in a mediocre local minimum. Always compare the converged value against a known reference (Hartree-Fock, MP2, classical CCSD).

3. Pay attention to optimizer-specific basin preferences. Different optimizers find different local minima from the same starting point, and their preferences can be exploited (or accidentally encode bias).

These are not theorems. They are tactical observations from practice. Most published VQA results follow them implicitly. Most failed VQA experiments did not.

Structural Role

This node sharpens the topology vocabulary of node 3.2 by zooming in on minima specifically. It is also the conceptual setup for Module 4 (barren plateaus): the failure mode introduced there is "all local minima become unreachable in expectation," which is a more extreme form of the problem characterized here.

Relations to Other Concepts

- **Loss landscape geometry of neural networks** — many of the same phenomena (many minima, mostly equivalent for overparameterized networks; basin connectivity).
- **Random energy models** — statistical physics models for random landscapes that predict the distribution of local minima energies.
- **Combinatorial optimization** — global vs local minima as a problem class.
- **Multi-start optimization** — the standard practical technique for landscape exploration.

Worked Example — Counting Minima In An HEA

Consider a 2-qubit HEA with 1 layer: $|\psi(\boldsymbol{\theta})\rangle = \text{CNOT} \cdot (R_Y(\theta_1) \otimes R_Y(\theta_2)) \cdot \text{CNOT} \cdot (R_Y(\theta_3) \otimes R_Y(\theta_4))|00\rangle$.

For a Hamiltonian $H = Z_0 + Z_1$, the cost function $C(\boldsymbol{\theta})$ is a 4D function. Numerical exploration shows:

- **At least 16 local minima** in the fundamental cell $[-\pi, \pi]^4$, arising from sign-flip symmetries: $(\theta_i \to -\theta_i + 2\pi)$ gives an equivalent state up to a global phase, and 4 parameters $\times$ 2 sign choices $= 2^4 = 16$.
- **The global minimum** is at $(0, 0, \pi, \pi)$ or its sign-flip equivalents, where both qubits are flipped to $|11\rangle$ and $\langle Z_0 + Z_1 \rangle = -2$.

- **All 16 minima have the same energy** (because they all map to the same state up to a global phase).
- **Saddles between them** correspond to half-flipped configurations with energy 0 or near-0.

So this 4-parameter ansatz has 16 equivalent global minima and a complicated structure of saddles between them. A gradient-descent optimizer starting from a random init will find one of the 16 — which one depends on the basin of attraction the init falls into — but since they're all equivalent, "finding the global minimum" is trivial here.

For a more interesting Hamiltonian (e.g., $H = Z_0 + Z_1 + 0.3X_0X_1$), the symmetry is broken slightly and the 16 minima split into 16 *near-equivalent* minima at slightly different energies. Then the choice of optimizer starts to matter: a finer optimizer will find a slightly lower one, but they're all close.

This is the pattern: **HEAs generate many minima by symmetry, and which one you find usually doesn't matter much — until you turn off the symmetry, at which point the structure suddenly becomes important.**

Visualization

Pending. A 2D slice of the 4D HEA landscape, showing 4 of the 16 minima (one quadrant of the sign-flip group). Color: cost. Annotations: basins, saddles, the global value. Caption: "4 of 16 equivalent minima."

Exercises

1. For the cost function $C(\theta_1, \theta_2) = -\cos\theta_1 - \cos\theta_2$ on $[-\pi, \pi]^2$, find all local minima. How many are there? How many are global?
2. Show that the cost function in Exercise 1 has the same number of local maxima as local minima, by Morse-theoretic reasoning.
3. (VQA) Describe a multi-start optimization protocol for a 20-parameter VQA. How many restarts would you do? How would you decide when to stop?

Research Extensions

- **Counting critical points of variational ansätze** — analytic upper and lower bounds.
- **Energy distribution of local minima** — what is the typical gap between the lowest local minimum found and the global minimum?
- **Basin volume estimation** — for a given local minimum, how big is its basin? Determines how easy it is to find by random initialization.
- **Saddle-aware optimizers** — methods that explicitly seek out and cross saddles.

Cross-links to Other Courses

- **Optimization Theory** — local vs global minima, convergence guarantees.
- **Statistical Physics** — random landscape models, Kac-Rice formula for critical point counting.
- **Module 4 (Barren Plateaus)** — the "all minima unreachable" failure mode.
- **Module 6 (Optimization Dynamics)** — how different optimizers handle multi-minima landscapes.

Symmetry in Landscapes

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.4

Symmetry in a VQA cost landscape is both a **gift** and a **curse**, depending on how you look at it.

The **gift**: a symmetric landscape has equivalent critical points related by group actions. Once you find one, you've effectively found them all. Symmetries can be exploited to restrict the search space.

The **curse**: symmetric landscapes have *redundant* parameter directions. Two distinct parameter vectors can produce the same physical state, which means the parameter-space gradient has a kernel — there are directions in parameter space that don't change anything, but the optimizer still tries to move along them, wasting effort.

This node walks through both sides of the symmetry coin and explains how to think about symmetry-aware optimization.

Three Sources of Symmetry

There are three distinct kinds of symmetry that show up in VQA cost functions, and they have very different consequences.

1. Hamiltonian Symmetries

If the Hamiltonian H commutes with some unitary G, then the eigenstates of H split into invariant subspaces under G, and the ground state can be chosen to lie in a specific symmetry sector.

Examples: - **Particle number symmetry**: $[H, \hat{N}] = 0$ for fermionic Hamiltonians. - **Total spin / spin projection**: for spin-conserving Hamiltonians. - **Translation invariance**: for lattice Hamiltonians on translation-invariant lattices. - **Parity** (Z2 symmetries): for many quantum chemistry Hamiltonians after Jordan-Wigner. - **Time-reversal**: for Hamiltonians made of real combinations of Pauli operators.

Consequences for the cost landscape: - The ground state lies in one specific symmetry sector. If the ansatz prepares states outside that sector, those states have higher energy than they "should" — the cost function has local minima outside the relevant sector that the optimizer might mistakenly find. - Symmetry-preserving ansätze (constructed to commute with G) restrict the search to the right sector, removing those spurious minima. - Symmetry-violating ansätze must "discover" the symmetry through optimization, which is generally slow and unreliable.

2. Ansatz Symmetries

If the ansatz $U(\boldsymbol{\theta})$ has a structural symmetry — e.g., parameters θ_i and θ_j are interchangeable — then the cost function inherits a **discrete symmetry group action** on parameter space.

Examples: - **Permutation symmetry of parallel layers**: in an HEA where every layer is identical, swapping the parameters of layer 3 with layer 5 produces a different parameter vector with the same cost. - **Sign-flip symmetry**: $R_Y(\theta)$ and $R_Y(-\theta + 2\pi)$ prepare states differing by a phase. This can be a 1- or 2-element symmetry depending on the surrounding circuit. - **Qubit permutation symmetry**: when the device topology and ansatz are symmetric under qubit relabeling.

Consequences for the cost landscape: - Critical points come in equivalence classes. A local minimum at $\boldsymbol{\theta}^*$ is also a local minimum at every $g \cdot \boldsymbol{\theta}^*$ for $g \in G$. - The number of critical points is multiplied by $|G|$. - Symmetry-related basins are equally good — the optimizer is indifferent between them.

This is the source of the "16 equivalent minima" phenomenon described in node 3.3.

3. Parameterization Redundancy (Continuous Gauge)

Some ansatz parameterizations have **continuous redundancies** — directions in parameter space that don't change the prepared state at all.

Example: $R_X(\theta_1)R_X(\theta_2)|0\rangle$ depends only on the sum $\theta_1 + \theta_2$, not on the individual values. The direction $(\theta_1, \theta_2) \to (\theta_1 + s, \theta_2 - s)$ is a 1-parameter gauge orbit — no physical motion, just relabeling.

Consequences for the cost landscape: - The cost function has flat directions along the gauge orbits. - The Hessian has a kernel; the metric g_{ij} is degenerate (rank deficient). - Any local minimum is part of an entire submanifold of equivalent minima of dimension equal to the gauge orbit dimension. - The natural gradient is undefined in the gauge directions.

These continuous redundancies are pathological — they waste parameters and slow optimization without producing any genuine minimum structure. Good ansatz design avoids them where possible.

How To Use Symmetry

The good move is: **identify symmetries and exploit them to shrink the search space.**

For Hamiltonian symmetries:

- **Restrict the ansatz** to states obeying the symmetry. For particle number, this means using particle-number-preserving gates. For parity, this means using only even Pauli rotations.
- **Encode the symmetry into the cost function** as a penalty. Add $\mu(\langle G\rangle - g_0)^2$ to the energy expectation, where g_0 is the desired symmetry charge. Pushes the optimizer toward the correct sector.
- **Project the state** onto the symmetric subspace before measurement. Equivalent to ground-state preparation in a smaller Hilbert space.

For ansatz symmetries:

- **Quotient out the symmetry** — use a parameterization where redundant directions are explicitly removed. Sometimes possible by clever ansatz design.
- **Symmetry-aware initialization** — start the optimizer at a representative point of each equivalence class to ensure broad exploration.
- **Symmetry-aware aggregation** — when reporting "the converged parameters," report the equivalence class rather than a specific representative.

For continuous gauge:

- **Eliminate the gauge** by reparameterizing. $R_X(\theta_1)R_X(\theta_2) \to R_X(\theta_1 + \theta_2)$.
- If you can't eliminate it (gauges arising from circuit composition aren't always removable), use the **natural gradient** (Module 6) which automatically projects out gauge directions.

How Symmetry Misleads

The bad move is to **ignore symmetry**. Three failure modes:

1. Optimizer wastes effort on gauge. A vanilla gradient descent on a gauge-redundant parameterization spends real shots evaluating the gradient in gauge directions, where it always reads zero plus noise. The optimizer is "moving" in those directions but the state isn't.

2. Symmetry-broken minima look like progress. If the cost function has a Hamiltonian symmetry but the ansatz doesn't preserve it, the optimizer will find local minima at symmetry-broken states. These look like "good progress" (the cost is decreasing) but the converged state has the wrong quantum numbers — it is not the physical ground state.

3. Spurious basins from broken parameterization symmetries. If your ansatz has a discrete symmetry the cost function does not, then the symmetry-related "minima" are not actually equivalent — they have slightly different costs because of the asymmetry — and the optimizer can get confused about which one to commit to.

The right diagnostic for all of these is to **inspect the converged state**, not just the converged cost. Compute the expectation values of the symmetry generators. If they don't match the expected charges, you have a symmetry-related bug.

Symmetry and Module 5

Module 5's regularization framework has a clean interaction with symmetry:

- **Prior injection (objective-side)** can encode symmetry as a penalty term added to the cost.
- **Dynamical modification (dynamics-side)** can encode symmetry as a constraint on the optimizer's update rule (project the gradient onto the symmetric subspace).

These produce *different* converged states even when nominally enforcing the same symmetry. The thesis's argument that these two regularization styles have different semantics applies in the symmetry case as well — perhaps especially clearly there.

Structural Role

This is the **structural awareness** node of Module 3. It rounds out the topology discussion (3.2-3.3) with the observation that not all topology is independent — structural relations between minima exist and can be exploited.

Forwards: every symmetry-aware ansatz design choice in Module 2.3 (chemically-inspired ansätze) is consistent with this node.

Relations to Other Concepts

- **Group theory** — the mathematical setting for discrete symmetries.
- **Gauge theory** — the formal treatment of continuous parameterization redundancies.
- **Equivariant neural networks** — the classical ML analogue of symmetry-preserving ansätze.
- **Symmetry breaking in physics** — the process by which a symmetric Hamiltonian acquires a non-symmetric ground state. Different phenomenon, often confused.
- **Quotient space optimization** — optimizing on the quotient by a symmetry group.

Worked Example — Sign-Flip Symmetry of R_Y

Consider $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle$.

For Hamiltonian $H = X$, the cost is

$$C(\theta) = \langle\psi|X|\psi\rangle = 2\cos(\theta/2)\sin(\theta/2) = \sin(\theta).$$

This has minima at $\theta = -\pi/2$ and $\theta = 3\pi/2$, and these differ by 2π — they are the **same point on the torus** S^1. One minimum, no symmetry.

Now consider $H = Z$. The cost is

$$C(\theta) = \cos(\theta).$$

Minimum at $\theta = \pi$. Maximum at $\theta = 0$. The function has reflection symmetry $\theta \to -\theta$, but $-\pi = \pi$ on the torus, so the symmetry has just one fixed point and no equivalence class structure. Again one minimum.

Now consider $H = X^2 + Z^2 = 2I$ (a useless Hamiltonian, but pedagogically clean): the cost is constant. Every point is equivalent. The "symmetry" is the entire group acting trivially.

These three examples show that the *manifest* symmetry of the cost function depends jointly on the ansatz and the Hamiltonian. **Symmetry isn't a property of the ansatz alone or the Hamiltonian alone; it's a property of their composition.**

Visualization

Pending. A 2D parameter space colored by cost, with discrete symmetry orbits highlighted (e.g., 4-fold permutation orbits). Multiple equivalent minima shown connected by group action arrows. Caption: "Symmetry × cost function = orbit structure."

Exercises

1. The Hamiltonian $H = Z_0 + Z_1$ is invariant under qubit swap. The ansatz $R_Y(\theta_1) \otimes R_Y(\theta_2)|00\rangle$ is also invariant under qubit swap. Show that the cost function has a $\mathbb{Z}_2$ symmetry, and characterize the orbits.
2. Design a symmetry-violating perturbation to the above Hamiltonian and show that the symmetry of the cost function is broken.
3. (VQA) Explain why a symmetry-preserving ansatz can sometimes have **fewer** local minima than a symmetry-breaking one. Is fewer minima always good?

Research Extensions

- **Equivariant VQAs** — ansätze whose action commutes with a target symmetry group by construction.
- **Symmetry-protected initial states** — restricting the reference state to lie in a specific symmetry sector.
- **Symmetry detection algorithms** — automatic discovery of symmetries in arbitrary Hamiltonians.
- **Quotient-space optimization** — optimizing on the parameter space modulo the symmetry group.

Cross-links to Other Courses

- **Group Theory** — discrete and continuous symmetries.
- **Differential Geometry** — quotient manifolds, principal bundles.
- **Equivariant ML** — neural networks invariant under group actions.
- **Module 2.3** — chemically-inspired ansätze are symmetry-aware by construction.

Landscape Geometry and Gradients

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.5

This node is about the **local** structure of a VQA cost landscape: gradient magnitude, curvature, the metric, and how these properties change as you move around parameter space.

Local geometry is what every optimizer "sees" — gradient descent uses the gradient, second-order methods use the Hessian, natural gradient uses the metric. Understanding local geometry is therefore the bridge between the topological vocabulary of nodes 3.2-3.4 and the operational behavior of an optimizer in Modules 5-7.

The Gradient

The cost function gradient is

$$\nabla C(\boldsymbol{\theta}) = \left(\frac{\partial C}{\partial \theta_1}, \ldots, \frac{\partial C}{\partial \theta_P} \right).$$

For a VQA with cost $C(\boldsymbol{\theta}) = \langle \psi(\boldsymbol{\theta}) | H | \psi(\boldsymbol{\theta}) \rangle$, each component is

$$\frac{\partial C}{\partial \theta_i} = \langle \partial_i \psi | H | \psi \rangle + \langle \psi | H | \partial_i \psi \rangle = 2 \, \mathrm{Re} \langle \partial_i \psi | H | \psi \rangle,$$

where $|\partial_i \psi\rangle$ is the partial derivative of the state with respect to θ_i.

For circuit ansätze built from Pauli rotations $R_P(\theta) = e^{-i\theta P/2}$, this can be evaluated **exactly** using the parameter-shift rule (node 3.8):

$$\frac{\partial C}{\partial \theta_i} = \frac{1}{2} \left[C(\boldsymbol{\theta} + \tfrac{\pi}{2} \mathbf{e}_i) - C(\boldsymbol{\theta} - \tfrac{\pi}{2} \mathbf{e}_i) \right].$$

This identity is what makes gradient-based VQA optimization viable — exact gradients can be computed by *evaluating the cost function* at shifted parameter values, no finite-difference approximation needed.

The trade is that each gradient component costs two cost evaluations. For a P-parameter ansatz, computing the full gradient costs $2P$ cost evaluations — and each cost evaluation is itself M measurement bases $\times$ N shots.

Gradient Magnitude

How big is $\|\nabla C(\boldsymbol{\theta})\|$ at a typical point?

For shallow problem-inspired ansätze (UCCSD, HVA, ADAPT-VQE on chemistry instances), the gradient magnitude at the Hartree-Fock starting point is usually substantial — large enough to drive meaningful descent.

For random initializations of expressive HEAs, the gradient magnitude is *exponentially small in qubit count* — this is the barren plateau result, which Module 4 will treat in full. At $n = 20$ qubits, the typical gradient magnitude in a random HEA region can be 10^{-6} or smaller, far below the shot-noise floor of any reasonable budget.

At critical points, gradient is exactly zero by definition. Near a critical point, the gradient grows linearly with distance from the critical point, with the rate set by the Hessian eigenvalues.

The practical question is: **is the gradient distinguishable from shot noise?** Shot noise sets a floor on detectable gradient magnitudes:

$$\text{detectable iff} \quad \|\nabla C\| \gtrsim \frac{\sigma_{\text{shot}}}{\sqrt{\text{shots/component}}}.$$

When the gradient magnitude drops below this threshold, the optimizer effectively cannot distinguish "no signal" from "small signal" and progress stalls. This is the **shot noise floor** that Module 5 spends a lot of effort working around.

The Hessian

The second-order structure is captured by the Hessian:

$$\mathbf{H}_{ij}(\boldsymbol{\theta}) = \frac{\partial^2 C}{\partial\theta_i \partial\theta_j}.$$

For Pauli-rotation ansätze, the Hessian can also be computed via parameter shifts (a "double-shift rule"), but each component costs four cost evaluations, so the full $P \times P$ Hessian costs $O(P^2)$ cost evaluations. For $P = 50$, that's 10,000 cost evaluations per Hessian — generally prohibitive.

The Hessian eigenvalues at a critical point classify it:

- **All eigenvalues positive**: local minimum.
- **All negative**: local maximum.
- **Mixed signs**: saddle point. The number of negative eigenvalues is the **Morse index** of the saddle.
- **Some zero eigenvalues**: degenerate critical point. Often signals a symmetry or gauge direction.

The **largest eigenvalue** of the Hessian is the **local Lipschitz constant** of the gradient and bounds how large a gradient-descent step you can take without overshooting.

The **condition number** of the Hessian (ratio of largest to smallest eigenvalue) determines convergence rate of gradient descent — large condition numbers mean slow convergence in the worst direction.

The Quantum Fisher Metric

The natural metric on the ansatz manifold is the **quantum Fisher information matrix** (QFIM):

$$g_{ij}(\boldsymbol{\theta}) = 4 \cdot \mathrm{Re}\left[\langle\partial_i\psi|\partial_j\psi\rangle - \langle\partial_i\psi|\psi\rangle\langle\psi|\partial_j\psi\rangle\right].$$

This is the **Fubini-Study metric** pulled back through the parameterization, scaled to make the Cramér-Rao bound clean.

The QFIM measures how distinguishable nearby parameter values are in terms of any possible quantum measurement. Two parameter directions with small g_{ii} produce nearly indistinguishable states; two with large g_{ii} produce very different states.

Crucially, the QFIM is **distinct from the Hessian of the cost function**. The Hessian depends on the cost function (and therefore on the Hamiltonian); the QFIM depends only on the ansatz.

The natural gradient update rule (Module 6) uses the QFIM to rescale the cost gradient:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\, g^{-1}(\boldsymbol{\theta}_k)\, \nabla C(\boldsymbol{\theta}_k).$$

This produces updates that are *invariant under reparameterization*: changing the way you label parameters doesn't change the trajectory in state space. Vanilla gradient descent does not have this property and can be wildly different under different parameterizations of the same ansatz.

Local Curvature and Step Size

The relationship between gradient magnitude and Hessian curvature determines the appropriate step size for gradient descent.

For a quadratic approximation $C(\boldsymbol{\theta}_k + \mathbf{d}) \approx C(\boldsymbol{\theta}_k) + \nabla C^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \mathbf{H} \mathbf{d}$, the optimal step in the gradient direction is

$$\eta^* = \frac{\|\nabla C\|^2}{\nabla C^T \mathbf{H} \nabla C}.$$

If you take a smaller step you make less progress; if you take a larger step you may overshoot. Adaptive learning rate methods (Adam, RMSProp) approximate this implicitly by scaling each component by an estimate of recent gradient variance, which is correlated with curvature.

In VQA practice, **the Hessian varies significantly across parameter space**, so a single global step size is rarely optimal. This is one reason adaptive methods often outperform vanilla gradient descent on VQAs.

Why Local Geometry Doesn't Tell You Everything

A subtle but important point: **local geometry is not sufficient to navigate a non-convex landscape.**

Even with perfect gradient and Hessian information at every point, a local optimizer: - Cannot tell whether the current basin is the global one. - Cannot find escape routes to better basins. - Cannot distinguish a long valley from a saddle. - Cannot detect concentration plateaus (Module 4) until it's already in one.

This is why **global** information — coming from population methods, surrogate models, or symbolic analysis — is often needed for hard VQA instances. Local geometry is the optimizer's microscope; it tells you the immediate slope but not the overall topology.

The thesis's choice of HOPSO as a representative population-based optimizer (Module 4-5 of the thesis) is motivated in part by this observation: HOPSO maintains a population of points that collectively sample the global structure, while each individual member follows local gradient information.

Structural Role

This node is the **local-structure** node of Module 3. It connects the topological vocabulary (3.2-3.4) to the gradient-computation machinery (3.8) and forwards to the statistical phenomena (3.6, 3.7) and barren plateaus (Module 4).

It is also the place where the QFIM is introduced informally; Module 6 will give it the full formal treatment as the source of the natural gradient.

Relations to Other Concepts

- **Calculus of variations** — first- and second-order necessary conditions for optimality.
- **Information geometry** — the Fisher metric and its quantum analogue.
- **Riemannian optimization** — optimizing on manifolds with non-trivial metrics.
- **Lipschitz constants** — the bound on gradient change that controls step size.
- **Newton's method** — second-order optimization that uses the inverse Hessian.

Worked Example — Gradient Of The Single-Qubit $H = Z$ Cost

For $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$ and $H = Z$:

$$C(\theta) = \cos(\theta), \quad C'(\theta) = -\sin(\theta), \quad C''(\theta) = -\cos(\theta).$$

At $\theta = 0$: $C = 1$, $C' = 0$, $C'' = -1$. Local maximum. At $\theta = \pi$: $C = -1$, $C' = 0$, $C'' = 1$. Local minimum (also global). At $\theta = \pi/2$: $C = 0$, $C' = -1$, $C'' = 0$. Inflection point.

Gradient magnitude at $\theta = \pi/2$ is 1, the largest possible. At $\theta = 0$ and $\theta = \pi$ it is 0.

The Hessian eigenvalue at $\theta = \pi$ is $+1$, so the local Lipschitz constant of the gradient is 1, and gradient descent with step size $\eta < 1$ is guaranteed to converge near this minimum.

For shot-noisy estimation with $N = 100$ shots and $\mathrm{Var}[\widehat{Z}] = 1 - \cos^2(\theta) = \sin^2(\theta)$, the noise floor is $\sin(\theta)/\sqrt{100} = 0.1\sin(\theta)$. At θ near π, this floor approaches 0 — the noise vanishes near the eigenstate. But the gradient also vanishes there, so the SNR doesn't necessarily improve.

This is a clean toy example of the relationship between gradient magnitude, curvature, and shot noise. **The optimizer is happy near the minimum because both signal and noise are small in absolute terms — what matters is their ratio.**

Visualization

Pending. A 1D plot of $C(\theta) = \cos(\theta)$ over $[-\pi, \pi]$, with overlay annotations: gradient arrow at $\theta = \pi/2$, curvature label at $\theta = \pi$, noise band around the curve. Caption: "Gradient and curvature, the optimizer's microscope."

Exercises

1. For $C(\theta_1, \theta_2) = \cos(\theta_1)\cos(\theta_2)$, compute the Hessian at the four critical points $(0,0), (0,\pi), (\pi,0), (\pi,\pi)$. Classify each.
2. For the same function, compute the QFIM g_{ij} for the parameterization $R_Y(\theta_1) \otimes R_Y(\theta_2)|00\rangle$. Compare to the Hessian of C. Are they the same? Why or why not?
3. (VQA) Estimate the typical gradient magnitude in a random HEA with 10 qubits and 5 layers from the Pauli-shift formula (without computing the full barren plateau theorem). Compare to the shot noise floor at $N = 1024$ shots per evaluation.

Research Extensions

- **Curvature-aware schedules** — using local Hessian information to adapt the step size online.
- **Approximate natural gradient** — block-diagonal or low-rank approximations to the QFIM that are cheap to compute.

- **Hessian estimation via parameter shifts** — efficient protocols that don't require $O(P^2)$ circuit runs.
- **Landscape topography from local probes** — reconstructing the global topology from local gradient/Hessian samples.

Cross-links to Other Courses

- **Calculus / Differential Geometry** — gradients, Hessians, Riemannian metrics.
- **Information Geometry** — Fisher information, quantum Fisher information.
- **Numerical Optimization** — Newton's method, quasi-Newton, trust regions.
- **Module 6 (Optimization Dynamics)** — the natural gradient and other geometric methods.

Concentration of Measure

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.6

This node is the **statistical** view of VQA cost landscapes. Where node 3.5 looked at the gradient at a single point, this node asks: **what does the gradient look like across all points, on average?**

The answer reveals one of the fundamental obstacles to scalable VQA optimization: in high-dimensional Hilbert spaces, almost all states look the same to almost all observables. This phenomenon is called **concentration of measure** and it is the deeper mathematical reason behind barren plateaus, the difficulty of random initialization, and several other VQA pathologies.

This node introduces concentration without yet doing the full barren plateau theorem (Module 4). The point here is to make the *general* phenomenon intuitive, so that the *specific* result in Module 4 lands as an instance of a broader pattern.

What Concentration Means

Concentration of measure is the observation that, in high-dimensional spaces with appropriate random structure, smooth functions take values very close to their mean almost everywhere.

The classical version: if $f : S^{n-1} \to \mathbb{R}$ is a 1-Lipschitz function on the unit sphere in $\mathbb{R}^n$, then for a uniformly random point $\mathbf{x}$ on the sphere,

$$\mathbb{P}\big[|f(\mathbf{x}) - \mathbb{E}[f]| > t\big] \leq 2\exp(-cnt^2).$$

For large n, this probability is very small unless t is very small. **Almost all of the sphere is mapped to a tiny window around the mean.**

This is counterintuitive because in low dimensions ($n = 2, 3$) it doesn't happen much — a sphere in 3D has plenty of room for a function to take varied values. But as n grows, the volume of the sphere increasingly concentrates near any equator, and any smooth function "averages out" over that volume.

The Quantum Version

For an n-qubit Hilbert space, the unit sphere is $S^{2\cdot 2^n - 1}$ — a sphere of dimension $2 \cdot 2^n - 1$, exponentially large in n.

If you draw a random pure state $|\psi\rangle$ from the **Haar measure** (the unique unitarily invariant probability measure on this sphere), then for any observable O with eigenvalue range $[\lambda_{\min}, \lambda_{\max}]$:

$$\mathbb{P}_{\psi\sim\text{Haar}}\big[|\langle\psi|O|\psi\rangle - \text{Tr}(O)/2^n| > t\big] \leq 2\exp(-c \cdot 2^n \cdot t^2/(\lambda_{\max} - \lambda_{\min})^2).$$

That is: **a Haar-random quantum state has expectation value almost exactly equal to** $\text{Tr}(O)/2^n$ **for almost any observable**, with concentration that gets exponentially tight as the qubit count grows.

For traceless observables like Pauli strings, $\text{Tr}(O)/2^n = 0$, so:

> A Haar-random state on n qubits has $\langle P\rangle \approx 0$ for every Pauli string P, with the typical deviation scaling as $2^{-n/2}$.

This is the *raw* concentration phenomenon for quantum states.

How Concentration Becomes a Cost Landscape Problem

When you put concentration together with the structure of an expressive ansatz, you get:

Step 1: A sufficiently deep, sufficiently random ansatz produces an approximate **2-design** — its parameter distribution induces a state distribution that approximates Haar measure on the relevant subspace.

Step 2: Therefore, for a typical random parameter vector $\boldsymbol{\theta}$, the expectation value $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ is concentrated near its mean $\mathrm{Tr}(H)/2^n$, with deviation $O(2^{-n/2})$.

Step 3: Variations in the parameter vector also produce variations in the cost — but those variations are also subject to concentration. The directional derivative of the cost is *also* a random variable concentrated near zero.

Step 4: Quantitatively, the gradient variance scales as $O(2^{-2n})$ (and the gradient norm as $O(2^{-n})$). This is the **barren plateau result** in its standard form.

Step 5: An optimizer drawing samples from this landscape sees gradients that are below the shot-noise floor for any reasonable budget, and progress stalls.

The barren plateau is concentration of measure made operationally lethal.

What Doesn't Concentrate

Not every ansatz produces concentrated gradients. The key escape routes are:

1. Restricted manifolds. If the ansatz only reaches a small subspace of Hilbert space (e.g., particle-number-conserving states for a fermionic Hamiltonian), the relevant measure is on that subspace, not the full sphere. The exponent in the concentration bound becomes |subspace| rather than 2^n, and concentration is much milder. UCCSD avoids barren plateaus precisely for this reason.

2. Local cost functions. If the cost is built from observables acting on $O(\log n)$ qubits, the concentration scales differently — the relevant Hilbert space is the small subsystem, and the concentration penalty is mild. Cerezo et al. (2021) proved that local cost functions admit gradients of order $\mathrm{poly}(1/n)$ rather than $O(2^{-n})$.

3. Shallow circuits. A circuit shallower than $O(\log n)$ cannot scramble enough to approximate a 2-design. Its state distribution is *structured*, not Haar-uniform, and the gradient does not concentrate as severely. Module 4 examines the threshold depth precisely.

4. Non-random initialization. If you start the optimizer at a non-random parameter (e.g., a Hartree-Fock-equivalent point or a warm start from a classical surrogate), you can avoid the random regime entirely. The barren plateau is a property of the *typical* point, not every point.

These escape routes are not always available, but when they are, they are the right way to design around the problem.

A Useful Mental Picture

Imagine an exponentially large sphere in some abstract space. On that sphere, almost all of the area is concentrated near any "equator" — a great-circle slice.

Now imagine you parameterize a curve on the sphere (the ansatz manifold) and drop a random point. With overwhelming probability, that point is near the equator. Worse, the curve's tangent at that point is nearly parallel to the equator, so moving along the curve doesn't move you away from the equator.

If your cost function is $f(\mathbf{x}) =$ signed distance from the equator, then: - The cost at a random point is near zero (concentration). - The gradient of the cost in the direction of curve motion is near zero (because the tangent is parallel to the equator). - All of this gets exponentially worse as the dimension grows.

That is the geometric picture of a barren plateau. **It is not that the answer isn't there. It is that the answer is overwhelmed by the volume of "everything else."**

Concentration In Other Disciplines

The same phenomenon appears in many other fields under different names:

- **Random matrix theory**: spectral measures of random matrices concentrate around a deterministic limit (Wigner semicircle, Marchenko-Pastur).
- **Statistical mechanics**: thermodynamic quantities concentrate near their ensemble averages in the thermodynamic limit.
- **High-dimensional statistics**: empirical estimators concentrate around population values.
- **Deep learning**: the loss landscape of deep neural networks at random initialization exhibits concentration phenomena that affect trainability ("vanishing gradients").

The pattern is universal: **as dimension grows, randomness produces uniformity**. VQAs are subject to this in a particularly sharp way because Hilbert space dimension grows exponentially with qubit count.

Why This Is The Deep Reason Things Are Hard

If you internalize concentration of measure as the underlying phenomenon, you understand barren plateaus, the difficulty of random initialization, the importance of inductive bias, and the tradeoff between expressivity and trainability — all as instances of a single mathematical fact.

This is the kind of reframing that the thesis's "stochastic objective generator" abstraction (1.8) was setting up: the optimizer sees a noisy scalar function whose statistical properties are determined by deep structural facts about the ansatz and the Hamiltonian. **Concentration of measure is one of the most important of those facts.**

It is also why "just use a more clever optimizer" cannot fix a barren plateau: the concentration is in the *function*, not in the optimizer. No amount of optimizer cleverness can detect a signal that isn't there. The fix has to come from the ansatz side or the cost side.

Structural Role

This node is the **statistical foundation** for Module 4 (barren plateaus), and an important conceptual node in its own right. It also explains, *in advance*, why several of Module 5's regularization techniques work — they push the optimizer into regions of parameter space where the concentration is mild.

Relations to Other Concepts

- **Lévy's lemma** — the canonical concentration inequality for spheres.
- **Talagrand's inequality** — the metric-space generalization.
- **Haar measure on $U(d)$** — the unique unitarily invariant probability measure on the unitary group.
- **t-designs** — a discrete approximation to Haar measure that captures the lowest t moments.
- **Cramér-Rao bound** — the information-theoretic lower bound that concentration phenomena saturate.

Worked Example — Concentration On A 4-Qubit Sphere

Take $n = 4$ qubits, so the relevant unit sphere has dimension $2 \cdot 16 - 1 = 31$. Draw a Haar-random state $|\psi\rangle$ and compute $\langle Z_0 \rangle$.

The mean over Haar is zero ($\mathrm{Tr}(Z_0)/16 = 0$). The variance over Haar is

$$\mathrm{Var}_{\mathrm{Haar}}[\langle Z_0 \rangle] = \frac{1}{2^n + 1} = \frac{1}{17} \approx 0.059.$$

So the standard deviation is $\sqrt{0.059} \approx 0.24$. For $n = 10$, the variance becomes $1/1025 \approx 10^{-3}$ and the std dev is 0.03. For $n = 20$, variance 10^{-6}, std dev 10^{-3}. For $n = 30$, variance 10^{-9}, std dev 3×10^{-5}.

At 30 qubits, a Haar-random state is essentially indistinguishable from one with $\langle Z_0 \rangle = 0$ using fewer than 10^9 shots. **The signal of any single Pauli expectation drowns in the volume of Hilbert space.**

This is the operational meaning of concentration: as the system size grows, you need exponentially many measurements to extract any structural information about a typical random state.

Visualization

Pending. Two-panel plot. Left: histogram of $\langle Z \rangle$ over many Haar-random states for $n = 2, 5, 10, 20$ qubits, showing the distribution narrowing dramatically. Right: log-log plot of $\mathrm{StdDev}[\langle Z \rangle]$ *vs n, showing exponential decay. Caption: "Concentration of measure: more qubits, less variation."*

Exercises

1. Show that for $n = 1$ (single qubit), the variance of $\langle Z \rangle$ over Haar-random states is $1/3$. (Hint: parameterize the state on the Bloch sphere and integrate.)
2. The concentration bound predicts variance $\sim 1/2^n$. Verify numerically by drawing Haar-random states for $n = 4, 6, 8$ qubits and computing the empirical variance of $\langle Z_0 \rangle$.
3. (VQA) Why does *local* cost (using only an observable on $O(\log n)$ qubits) avoid the worst of the concentration bound? Sketch the argument.

Research Extensions

- **Concentration in t-designs** — sharper concentration results for the actual distributions induced by random circuits.
- **Anti-concentration ansätze** — ansatz designs that provably avoid the concentration regime.
- **Initialization strategies** — distributions over parameters that avoid Haar-typical states.
- **Concentration in the presence of noise** — does noise help or hurt the concentration phenomenon?

Cross-links to Other Courses

- **Probability Theory** — concentration inequalities, Lévy's lemma.
- **Random Matrix Theory** — spectral concentration.
- **Information Theory** — typicality, asymptotic equipartition.
- **Module 4 (Barren Plateaus)** — the specific application to VQA gradients.

Shot Noise as Cost Stochasticity

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.7

So far in Module 3, the cost function has been treated as a *deterministic* scalar function on parameter space. This node introduces the **stochastic** view: in practice the optimizer never sees $C(\boldsymbol{\theta})$ — it sees a noisy estimate $\widehat{C}(\boldsymbol{\theta})$ corrupted by shot noise.

The relationship between C and $\widehat{C}$ is the relationship between **what the landscape really is** and **what the optimizer thinks it is**, and a lot of VQA practice consists of managing the gap between the two.

This node lays out the structure of that gap.

The Two Cost Functions

Let's be explicit.

True cost:

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle.$$

A deterministic function of $\boldsymbol{\theta}$. What you'd compute from a perfect statevector simulation.

Estimated cost:

$$\widehat{C}(\boldsymbol{\theta}) = \sum_j h_j \widehat{\langle P_j\rangle}, \quad \widehat{\langle P_j\rangle} = \frac{1}{N_j}\sum_{s=1}^{N_j} \widehat{e}_j^{(s)}.$$

A random variable. What the optimizer actually receives. $\mathbb{E}[\widehat{C}(\boldsymbol{\theta})] = C(\boldsymbol{\theta})$ (unbiased estimator, in the absence of hardware noise).

The optimizer's job is to find $\arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$, but its only access to C is through repeated samples of $\widehat{C}$. This is a stochastic optimization problem in the textbook sense.

The Variance Structure

The variance of $\widehat{C}(\boldsymbol{\theta})$ is

$$\mathrm{Var}[\widehat{C}(\boldsymbol{\theta})] = \sum_j \frac{h_j^2(1-\langle P_j\rangle^2)}{N_j}.$$

This formula has several important features:

1. Inverse in shot count. Doubling N_j halves the variance contribution from term j.

2. Proportional to coefficient squared. Terms with large $|h_j|$ contribute more variance per shot.

3. State-dependent. The factor $(1-\langle P_j\rangle^2)$ means that variance is *minimized* when $\langle P_j\rangle = \pm 1$ (the state is an eigenstate of P_j) and *maximized* when $\langle P_j\rangle = 0$. **The variance depends on $\boldsymbol{\theta}$.**

The third point is the most important: **the noise on $\widehat{C}$ is not uniform across parameter space**. Some regions are intrinsically noisier than others, and the optimizer experiences a *changing noise profile* as it moves around.

For chemistry Hamiltonians where the dominant Pauli terms have $h_j \sim 1$, the typical variance contribution per term is order 1 over N shots, and total variance is $\sum_j h_j^2/N$ — easily 0.01-1 for realistic settings. That's

a per-evaluation standard deviation of 0.1-1 hartree, which is large compared to chemical accuracy (~ 0.001 hartree).

Where The Variance Hides The Structure

When the variance of $\widehat{C}$ is large compared to the *local variation* of C, the optimizer cannot distinguish "small motion in $\boldsymbol{\theta}$" from "no motion in $\boldsymbol{\theta}$."

Define the **signal-to-noise ratio** at a point:

$$\mathrm{SNR}(\boldsymbol{\theta}) = \frac{\|\nabla C(\boldsymbol{\theta})\|}{\sqrt{\mathrm{Var}[\widehat{C}(\boldsymbol{\theta})]}}.$$

When SNR is large, the optimizer sees the gradient clearly and can descend. When SNR is near 1, the optimizer's view is dominated by noise and progress is slow. When SNR is much less than 1, the optimizer is essentially doing a random walk.

The dangerous regime is **near critical points and on plateaus**, where the gradient is small but the variance is not. This is exactly where you most want to make small careful steps, and exactly where you can't tell whether your steps are doing anything.

This is the **shot noise floor** that limits VQA optimization in practice. It is not addressed by being smarter; it requires either more shots, or a smaller variance contribution at the relevant points, or better signal extraction.

How Shot Noise Differs From Other Optimization Noise

Shot noise has properties that make it different from generic stochastic optimization noise:

1. State-dependent variance. Unlike the constant Gaussian noise often assumed in stochastic optimization theory, shot noise variance depends on the parameter point. Standard convergence analyses (e.g., Robbins-Monro) assume bounded variance and may not directly apply.

2. Bounded estimator outputs. Each shot gives a value in $[-1, +1]$ for each Pauli, so $\widehat{C}$ has bounded support. This is mostly a feature — sub-Gaussian concentration applies — but it means certain heavy-tailed-noise techniques are not relevant.

3. Correlated across the gradient. When you compute $\nabla\widehat{C}$ by parameter-shift, the same shots are used for shifted evaluations within a single component but independent shots are used across components. This produces a specific (block-diagonal) covariance structure that some optimizers can exploit and others can't.

4. Adaptive in principle. You can choose to spend more shots in some regions of parameter space than others — variance is not a fixed property of the problem, it is a budget allocation. Module 5's "shot-adaptive weighting" technique exploits this.

5. Time-stationary. On simulated hardware, shot noise is i.i.d. across iterations. On real hardware, it can drift due to calibration changes. Most analysis assumes the i.i.d. case.

These features are why VQA optimization is its own subfield rather than a special case of generic stochastic optimization.

Cost-Side vs Optimizer-Side Mitigation

The shot noise problem can be attacked from either side:

Cost-side: reduce the variance of $\widehat{C}$ at fixed shot budget.

- Better measurement grouping (node 2.8) → lower per-shot variance.
- Variance-weighted shot allocation → put shots where they matter most.
- Local cost functions → replace high-variance global observables with low-variance local ones.
- Importance sampling — query terms with large $|h_j|$ more often.

Optimizer-side: make the optimizer robust to whatever noise is there.

- Larger batch sizes (more shots per evaluation) → lower per-evaluation variance.
- Momentum / averaging methods (Adam, EMA gradients) → average over noise across iterations.
- Trust region methods — only commit to a step when the noise is small enough to be confident.
- Population methods (HOPSO) → collectively average over many points.

The thesis argues that *both* sides should be considered, and that the regularization techniques in Module 5 are best understood as the dynamics-side counterpart to the cost-side techniques here.

What Shot Noise Looks Like To The Optimizer

A toy mental model: imagine the true cost surface, then add a random Gaussian field of standard deviation $\sigma(\boldsymbol{\theta})$ on top of it. The Gaussian field is independent at each evaluation but has the same variance structure across evaluations (assuming i.i.d. noise).

What the optimizer sees at any single iteration is: - The true gradient direction, plus - A random gradient component proportional to the local noise level, plus - Anything from the optimizer's own update rule (momentum, etc.).

If the random component dominates the true gradient, the optimizer is essentially walking randomly. If the true gradient dominates, the optimizer is doing approximate gradient descent.

The transition between these regimes happens around SNR = 1, and the location of that transition in parameter space depends on the local landscape and the shot budget. **The optimizer crosses in and out of "useful" and "useless" regions during a single optimization run.**

Structural Role

This is the **stochasticity** node of Module 3 and the bridge to gradient computation (3.8) and to Module 5 (where the regularization framework is built around managing this exact problem).

It is also one of the cleanest places to see the "stochastic objective generator" abstraction (1.8) at work: the optimizer sees a noisy scalar oracle whose properties are determined by upstream physics. The shot noise is the dominant source of that noise on simulated hardware.

Relations to Other Concepts

- **Stochastic optimization theory** — Robbins-Monro, Kiefer-Wolfowitz, SGD, SPSA.
- **Sub-Gaussian random variables** — the formal class to which bounded shot noise belongs.
- **Variance reduction techniques** — control variates, stratified sampling, importance sampling.
- **Bayesian optimization** — explicitly models the stochastic oracle and uses Bayesian inference to decide where to query.
- **Robust optimization** — optimizing in the presence of bounded but unknown perturbations.

Worked Example — Variance At Three Parameter Points

For $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$ and $H = Z + 0.5X$:

$$C(\theta) = \cos(\theta) + 0.5\sin(\theta).$$

At $\theta = 0$ (state $|0\rangle$): $\langle Z\rangle = 1$, $\langle X\rangle = 0$. Cost = 1. Variance contributions: $(1-1) + 0.25(1-0) = 0.25$. Standard deviation = $0.5/\sqrt{N}$.

At $\theta = \pi$ (state $|1\rangle$): $\langle Z\rangle = -1$, $\langle X\rangle = 0$. Cost = -1. Variance: $0 + 0.25 = 0.25$. Same as above.

At $\theta = \pi/2$ (state $|+\rangle$): $\langle Z\rangle = 0$, $\langle X\rangle = 1$. Cost = 0.5. Variance: $(1-0) + 0.25(1-1) = 1$. Standard deviation = $1/\sqrt{N}$.

So the noise on $\widehat{C}$ varies by factor of 2 across parameter space. At eigenstate-like points (where one of the Pauli terms is saturated), variance is lower. At superposition points, variance is higher.

For an optimizer trying to detect a small gradient near $\theta = \pi/2$, the noise floor is *higher* there than near the minimum, even though the gradient is *also* higher. The SNR may or may not be favorable depending on the exact geometry.

This is the kind of effect that gets averaged away in textbook analyses but matters operationally.

Visualization

Pending. A 1D plot of $C(\theta)$ over $[-\pi, \pi]$, with a noise band of width $\sigma(\theta)$ around the curve, showing variable thickness. Annotation: "noise is wider at superposition points, narrower near eigenstates."

Exercises

1. For $H = Z_0 Z_1$ on 2 qubits and a state $|\psi\rangle = \cos(\theta/2)|00\rangle + \sin(\theta/2)|11\rangle$, compute $\langle ZZ\rangle(\theta)$ and $\mathrm{Var}[\langle ZZ\rangle](\theta)$. Where is the variance maximized?
2. Show that if a cost function is the expectation value of a single observable bounded in $[-1, +1]$, then $\mathrm{Var}[\widehat{C}] \leq 1/N$ where N is the shot count, regardless of the state.
3. (VQA) For an optimizer running with adaptive shot allocation, design a heuristic that increases shot count when SNR is low. What metric would you use to decide?

Research Extensions

- **Variance-aware shot allocation** — dynamically increasing shots in high-noise regions.
- **Importance-weighted estimators** — biasing measurements toward terms with large coefficients.
- **Variance penalization (Module 5.7)** — regularizing the cost to prefer low-variance regions.
- **Bandit-style stopping rules** — terminating evaluation when confidence intervals are tight enough.

Cross-links to Other Courses

- **Statistics** — sample variance, central limit theorem, sub-Gaussian concentration.
- **Stochastic Optimization** — Robbins-Monro, SPSA, Polyak averaging.

- **Module 5 (Regularization)** — the dynamics-side approach to managing this noise.
- **Module 9 (Hardware Noise Models)** — noise sources beyond shot noise.

The Parameter-Shift Rule

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Cost Landscapes **Node:** 3.8

This is the **closing node** of Module 3 and the place where the gradient computation question becomes concrete.

The parameter-shift rule is the small mathematical fact that makes gradient-based VQA optimization possible: **for ansätze built from Pauli rotations, the gradient of the cost function with respect to any parameter can be computed exactly by evaluating the cost function at two shifted parameter values**, with no need for finite differences or symbolic differentiation through the circuit.

This is a more remarkable result than it sounds. Without it, you would have to estimate gradients from finite differences (which compound shot noise) or differentiate through the entire quantum circuit symbolically (which is infeasible at scale). Parameter-shift gives you exact gradients with shot-statistics-only error — a clean separation that makes the rest of VQA optimization theory tractable.

The Statement

For a parameterized circuit where parameter θ_i appears in a single Pauli rotation gate $R_P(\theta_i) = e^{-i\theta_i P/2}$ (with P a Hermitian operator with eigenvalues ± 1), the gradient of any expectation value $C(\boldsymbol{\theta}) = \langle O \rangle_{\boldsymbol{\theta}}$ is

$$\frac{\partial C}{\partial \theta_i} = \frac{1}{2}\left[C\left(\boldsymbol{\theta} + \frac{\pi}{2}\mathbf{e}_i\right) - C\left(\boldsymbol{\theta} - \frac{\pi}{2}\mathbf{e}_i\right)\right].$$

This is **exact**, not an approximation. There is no $O(\Delta t^2)$ error like in finite differences. The only error in the estimated gradient is shot noise from estimating the two cost values.

The shifts are by exactly $\pi/2$ — not a small step, but a *symmetric* large step that exploits the trigonometric structure of the underlying functional dependence.

The Derivation

Why does this work?

The key fact is that for $P^2 = I$ (which holds for any Pauli string), the rotation gate has the simple expansion

$$R_P(\theta) = \cos(\theta/2)I - i\sin(\theta/2)P.$$

So if the cost function depends on θ_i only through this gate, it has the functional form

$$C(\theta_i) = a + b\cos(\theta_i) + c\sin(\theta_i)$$

for some θ_i-independent constants a, b, c. (Substituting in the rotation expansion and computing $\langle\psi(\theta_i)|O|\psi(\theta_i)\rangle$ gives precisely this form — the cosine and sine come from squaring the rotation amplitudes.)

A function of this form has derivative

$$C'(\theta_i) = -b\sin(\theta_i) + c\cos(\theta_i).$$

Now compute $C(\theta_i + \pi/2) - C(\theta_i - \pi/2)$:

$$C(\theta_i + \pi/2) = a + b\cos(\theta_i + \pi/2) + c\sin(\theta_i + \pi/2) = a - b\sin(\theta_i) + c\cos(\theta_i).$$
$$C(\theta_i - \pi/2) = a - b\cos(-\theta_i + \pi/2) + c\sin(-\theta_i + \pi/2)... = a + b\sin(\theta_i) - c\cos(\theta_i)...$$

(Working it out more carefully:)

$$C(\theta_i + \pi/2) - C(\theta_i - \pi/2) = 2(-b\sin(\theta_i) + c\cos(\theta_i)) = 2C'(\theta_i).$$

So $C'(\theta_i) = \frac{1}{2}[C(\theta_i + \pi/2) - C(\theta_i - \pi/2)]$. **The shift derivative is the exact derivative because the underlying function is a sinusoid.**

The derivation only used: 1. $P^2 = I$ (for the rotation expansion to be sinusoidal). 2. θ_i appears in exactly one place in the circuit.

Both conditions are true for standard Pauli-rotation ansätze, which is why parameter-shift works for them.

Where Parameter-Shift Doesn't Apply

Parameter-shift in its basic form has assumptions, and there are ansätze where it doesn't work.

1. Gates whose generators have more than two distinct eigenvalues. A gate of the form $e^{-i\theta G}$ where G has eigenvalues $\{\lambda_1, \lambda_2, \lambda_3, \dots\}$ gives a cost function that is a sum of multiple sinusoids with different frequencies, not a single sinusoid. The naive 2-shift rule doesn't apply, and you need a *generalized parameter-shift rule* with more shifts to disentangle the frequencies.

For example: a controlled rotation gate has a non-Pauli generator. The generalized rule for it uses 4 shifts.

2. Same parameter appearing in multiple gates. If θ_i appears in two different rotation gates, the function depends on θ_i through a sum of two sinusoids and the 2-shift rule again doesn't suffice.

3. Hardware-specific gate decompositions. If the device decomposes a "logical" $R_P(\theta)$ into a different physical gate sequence (e.g., $e^{-i\theta P/2} \to$ several pulses with θ-dependent durations), parameter-shift may not apply at the pulse level.

4. Non-unitary noise channels. If the circuit includes intermediate measurements or non-unitary operations, the simple algebraic structure breaks down.

For practical VQA work with standard ansätze (HEA, UCCSD, QAOA, HVA), parameter-shift in its basic form covers nearly all cases. The generalized rules are well-understood for the cases it doesn't.

Cost Comparison: Parameter-Shift vs Finite Difference

Parameter-shift: 2 cost evaluations per gradient component. Exact gradient (up to shot noise).

Forward finite difference: 2 cost evaluations per component (one shifted, one baseline). Approximation error $O(\Delta t)$, where Δt is the shift.

Central finite difference: 2 cost evaluations per component. Approximation error $O(\Delta t^2)$.

The **circuit-evaluation cost is the same** (2 evaluations per component) for parameter-shift and for either finite difference variant. The difference is in the **error**: parameter-shift has zero discretization error, while finite difference has $O(\Delta t)$ or $O(\Delta t^2)$ error.

For finite difference, you face the standard tradeoff: - Small Δt: low discretization error but small numerator ($C(\theta + \Delta t) - C(\theta)$ is small) which gets dominated by shot noise. - Large Δt: better signal-to-noise ratio in

the numerator but bigger discretization error. - Optimal Δt is somewhere in the middle and depends on the unknown gradient magnitude and noise level.

Parameter-shift sidesteps this entirely. The shift $\pi/2$ is large enough to give a well-conditioned numerator and produces zero discretization error regardless of the local function behavior. This is the operational reason it dominates finite difference for VQA gradient estimation.

What This Means For The Optimizer

A parameter-shift gradient evaluation for a P-parameter ansatz costs $2P$ cost function evaluations. At $P = 50$, that's 100 evaluations per gradient — and at, say, $M = 10$ Pauli measurement groups $\times$ $N = 1000$ shots per group, each cost evaluation is 10,000 circuit runs, so a single full gradient is **1 million circuit runs**.

This is *much* cheaper than computing the Hessian ($O(P^2)$ gradients = $O(P^3)$ circuit runs) but still substantial.

Three optimizer strategies emerge:

1. Full-gradient methods (BFGS, vanilla gradient descent): pay the $2P$ cost per iteration, take an informed step.

2. Subsampled-gradient methods (SPSA, random coordinate descent): pay only $O(1)$ or $O(\log P)$ cost per iteration but get a noisier gradient estimate. SPSA is famous for getting a good descent direction from just *one* parameter-shift pair, regardless of how many parameters there are.

3. Gradient-free methods (COBYLA, Bayesian optimization, HOPSO): skip parameter-shift entirely and work only from cost evaluations. Each iteration is cheap but informed updates require more iterations.

The choice between these is driven largely by per-evaluation cost — and per-evaluation cost is set by the parameter-shift rule's $2P$ overhead. **The parameter-shift rule is what makes gradient-based VQA optimization possible, and its cost is what makes gradient-free alternatives competitive.**

Closing Module 3

This is the last node of Module 3. With the parameter-shift rule in hand, you have:

- A vocabulary for the topology of cost landscapes (3.2-3.4).
- An understanding of local geometry and gradients (3.5).
- The statistical view (concentration, 3.6).
- The stochastic view (shot noise, 3.7).
- The operational mechanism for gradient computation (3.8, this node).

Module 4 will use all of these to prove and analyze the barren plateau theorem. Module 5 will use them to motivate regularization. Module 6 will use them to design optimization update rules. Modules 7-10 will return to them as needed.

The cost landscape is now fully described.

Structural Role

This is the operational closure of Module 3. It also forwards directly to Module 6, which builds optimizers that use parameter-shift gradients as their primary input.

Relations to Other Concepts

- **Numerical differentiation** — finite differences as the classical alternative to parameter-shift.
- **Automatic differentiation** — computational graph differentiation for general functions; parameter-shift is essentially a closed-form auto-diff for unitary circuits.
- **Generalized parameter-shift rules** — extensions to gates with non-Pauli generators (Wierichs et al. 2022).
- **Stochastic parameter shift** — variants that randomize the shift to avoid certain pathologies.

Worked Example — Verifying Parameter-Shift On A Single Qubit

For $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$ and $H = Z$, the cost is $C(\theta) = \cos(\theta)$, so $C'(\theta) = -\sin(\theta)$.

Let's verify the parameter-shift rule at $\theta = \pi/4$.

By calculus: $C'(\pi/4) = -\sin(\pi/4) = -1/\sqrt{2} \approx -0.707$.

By parameter-shift:

$$C(\pi/4 + \pi/2) - C(\pi/4 - \pi/2) = \cos(3\pi/4) - \cos(-\pi/4) = -1/\sqrt{2} - 1/\sqrt{2} = -2/\sqrt{2} = -\sqrt{2}.$$

$$\frac{1}{2}[C(\pi/4 + \pi/2) - C(\pi/4 - \pi/2)] = -\sqrt{2}/2 = -1/\sqrt{2} \approx -0.707.$$

Match. The parameter-shift rule gives exactly the analytic derivative. No finite-difference error, no truncation. Just two cost evaluations and an arithmetic combination.

For a real device, each of those two cost evaluations would be a full noisy estimation: prepare $R_Y(3\pi/4)|0\rangle$, measure Z, N shots; then prepare $R_Y(-\pi/4)|0\rangle$, measure Z, N shots; subtract the sample means; divide by 2. The result is an unbiased estimator of the exact gradient with variance $\sim 1/N$.

Visualization

Pending. A 1D plot of $C(\theta) = \cos(\theta)$ with two shifted points $(\theta - \pi/2, C(\theta - \pi/2))$ and $(\theta + \pi/2, C(\theta + \pi/2))$ marked, and a tangent line at θ whose slope is the parameter-shift gradient. Caption: "Two evaluations, exact slope."

Exercises

1. Verify the parameter-shift rule for the cost function $C(\theta) = \langle X \rangle = \sin(\theta)$ at $\theta = \pi/3$.
2. Show that for a function of the form $f(\theta) = a + b\cos(\theta) + c\sin(\theta) + d\cos(2\theta) + e\sin(2\theta)$, the simple 2-shift rule gives an *incorrect* derivative. Sketch the generalized rule (using 4 shifts) that recovers the correct derivative.
3. (VQA) For a 50-parameter ansatz, full-gradient evaluation costs 100 cost evaluations per iteration. SPSA estimates a gradient direction from just 2 cost evaluations per iteration. Sketch the conditions under which SPSA's per-iteration noise is better-traded against the full-gradient cost.

Research Extensions

- **Pulse-level parameter-shift** — extending parameter-shift to ansätze defined at the pulse level rather than the gate level.

- **Reduced-shot parameter-shift** — protocols that use fewer shots per gradient component by reusing measurements.
- **Stochastic parameter-shift** — randomized shifts that improve some statistical properties.
- **Higher-order shift rules** — direct estimation of Hessians via 4-shift rules.

Cross-links to Other Courses

- **Numerical Methods** — finite differences, automatic differentiation.
- **Calculus** — exact derivatives via trigonometric identities.
- **Module 6 (Optimization Dynamics)** — how this gradient is used by optimizers.
- **Module 1 (Node 1.4)** — earlier introduction of the parameter-shift rule from the architecture side.

Module 4 — Barren Plateaus

Gradient Vanishing in Quantum Circuits

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.1

This is the **opening node** of Module 4 — the module that takes the concentration-of-measure machinery from node 3.6 and the shot noise analysis from node 3.7 and welds them into the single most important *obstacle* in modern VQA practice: the **barren plateau**.

Before the theorem (node 4.2), we need the phenomenon. This node describes what gradient vanishing actually looks like in a quantum circuit — how an optimizer experiences it, how it was first noticed, and why it is qualitatively different from the "vanishing gradient" problem in classical deep learning that shares its name.

The Phenomenon

Run this experiment mentally:

1. Take a hardware-efficient ansatz with n qubits and L layers, where L is "deep enough" — say $L \geq n$.
2. Initialize the parameters uniformly at random in $[0, 2\pi)$.
3. Pick any traceless observable — say Z_0 on qubit 0 — and define the cost $C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|Z_0|\psi(\boldsymbol{\theta})\rangle$.
4. Compute $\partial C/\partial\theta_i$ for any single parameter θ_i, either analytically (via parameter-shift) or numerically.

You will find, with overwhelming probability, that the partial derivative is **exponentially small in the number of qubits**. At $n = 4$, it might be 10^{-1}. At $n = 10$, 10^{-3}. At $n = 20$, 10^{-6}. At $n = 30$, 10^{-9}.

The mean of the gradient is zero. The variance of the gradient is exponentially small. The **absolute** gradient magnitude therefore also scales exponentially in qubit count.

This is not a single-point pathology. It is the *typical* behavior across the vast majority of parameter space. Random initialization almost always lands somewhere with no detectable gradient signal.

What This Looks Like To The Optimizer

Imagine being a gradient-descent optimizer plopped into this landscape.

You start at $\boldsymbol{\theta}_0$ (random). You spend your shot budget estimating the cost and the gradient. Because the true gradient is $\sim 10^{-6}$ and your shot-noise floor is $\sim 10^{-2}$, the estimated gradient is pure noise. You "take a step" — but the step direction is random, uncorrelated with any meaningful descent direction. You re-evaluate the cost — it hasn't changed (or has changed by a random amount within shot noise). You take another step. Same result.

From the optimizer's point of view: **the cost is flat, everywhere you look, forever.** The optimizer has no way to distinguish "we're at a minimum" from "the signal is below the noise floor" from "there is no signal here." It simply never makes any progress, and no amount of optimizer cleverness can help.

This is the **operational** face of concentration of measure. Barren plateaus are not dangerous because cost is large or the landscape is ugly. They are dangerous because the signal has *left the building* — the cost function is, for all practical purposes, a constant function plus noise.

Historical Origin

The phenomenon was systematically documented by **McClean, Boixo, Smelyanskiy, Babbush, and Neven (2018)** in a paper titled "Barren plateaus in quantum neural network training landscapes." They ran numerical experiments on random hardware-efficient ansätze with increasing qubit counts, computed the variance of gradients, and observed the exponential decay. They also gave a first proof sketch showing that the variance is bounded above by an expression that decays as 2^{-2n} for deep-enough ansätze.

The follow-up theorem (node 4.2) makes this exponential decay a **statement about any sufficiently-expressive ansatz**, connecting it to t-design theory and Haar measure integration. The 2018 paper was, in one stroke, the moment when the VQA community realized that the naive recipe — "use a deep HEA, initialize randomly, optimize" — does not scale.

Before 2018, many VQA proposals assumed that adding more layers (more expressivity) would make optimization easier because the landscape would have richer structure. After 2018, the opposite became clear: adding more layers often makes optimization *harder*, because the landscape becomes more Haar-random and therefore more concentrated.

This was the single most important update in the field's understanding of what makes VQAs scale — or rather, what prevents them from scaling.

Vanishing Gradients: Quantum vs Classical Deep Learning

The phrase "vanishing gradients" was already well-known in classical deep learning before 2018, where it describes a different but related phenomenon:

- **Classical vanishing gradients** (Hochreiter 1991, resolved for RNNs via LSTMs): the gradient of the loss with respect to early-layer weights in a deep network vanishes because the chain rule repeatedly multiplies small Jacobians. The fix is architectural — use gates (LSTM, GRU), skip connections (ResNet), normalization (BatchNorm), or better initialization (Xavier, Kaiming).
- **Quantum barren plateaus**: the gradient of the cost with respect to *any* parameter vanishes because the cost function is a concentrated observable on a random state. The fix is not architectural in the same sense — no amount of "skip connections" or "normalization" addresses it, because the fundamental issue is that Hilbert space is exponentially large and high-dimensional random states look all the same.

The two phenomena share a symptom (gradients too small to optimize) but have different causes and different fixes. A key insight is that **classical fixes do not transfer directly**. You can't just plug an LSTM into a quantum circuit and expect barren plateaus to go away. The fix has to address the *concentration of measure* directly, which is typically done by restricting the ansatz expressivity (more structure, less randomness) rather than by adding more machinery on top.

Why The Naive Fix Doesn't Work

"Just use more shots."

This is the first thing you think of, and it does not work in any scalable sense. The shot noise floor scales as $1/\sqrt{N}$. To detect a gradient of magnitude 2^{-n} with SNR 1, you need $N \sim 2^{2n}$ shots per gradient component. For $n = 20$ qubits, that is $\sim 10^{12}$ shots per component, $\sim 10^{14}$ shots per full gradient. At realistic quantum hardware rates (say 10^3 shots/second per circuit), this is $\sim 10^{11}$ seconds of hardware time, or about 3,000 years per gradient evaluation.

The shot-count fix is exponentially expensive. It is not a strategy; it is a *refutation* of the idea that barren plateaus can be "just optimized around."

"Just use a smarter optimizer."

Also doesn't work. As argued in node 3.6, the concentration is in the *function* itself. No optimizer — SGD, Adam, natural gradient, SPSA, CMA-ES, Bayesian optimization — can extract signal that is below the noise floor. They can at best *not waste* the signal that is present, but they cannot create more signal.

"Use second-order information."

Also doesn't work. The Hessian is *also* concentrated (a straightforward extension of the gradient concentration proof), so second-order methods face the same exponential-scaling problem.

The conclusion: **barren plateaus are a structural property of the chosen ansatz and cost function combination, not a property of the optimizer.** You cannot fix them from the optimizer side. You must change the ansatz (restricted manifolds, locality, depth) or the cost (local observables) or the initialization (warm starts). These are the contents of node 4.6.

The Practical Signature of a Barren Plateau

How do you know you're in one in practice? A few diagnostic signatures:

1. **Training curve is flat from iteration 0.** The cost doesn't decrease. It doesn't increase. It oscillates within shot noise around its initial value.
2. **Gradient norm is below shot noise.** If you compute $\|\widehat{\nabla C}\|$ and compare to σ_{shot}, the ratio is ~ 1 or smaller. You're not getting useful gradient information.
3. **Multiple random restarts give the same flat behavior.** If 10 different random initializations all produce flat training curves, it's probably not bad luck — you're in the plateau regime.
4. **The same ansatz with fewer qubits trains fine.** A classic barren plateau signature: the optimization works at $n = 4$, degrades at $n = 8$, and fails entirely by $n = 12$. The exponential scaling is on display.
5. **The cost value is very close to** $\text{Tr}(H)/2^n$**.** This is the Haar-expected value of a typical random state. If you're hovering near this value and not moving, you're in the plateau.

The diagnostic is useful because the alternative — "bad optimizer setup" — produces different signatures (erratic trajectories, divergent behavior, occasional progress). The barren plateau has a *distinctive* quiet signature: nothing happens, consistently.

Structural Role

This node is the **phenomenology** entry point for Module 4. It provides the motivation for the theorem (4.2), introduces the vocabulary the rest of the module will use, and makes the case that barren plateaus are the central obstacle to scalable VQA.

Everything in Modules 5, 6, and 8 that is described as "dealing with barren plateaus" derives its motivation from the phenomena described here.

Relations to Other Concepts

- **Concentration of measure (3.6)** — the deeper mathematical mechanism.
- **Shot noise (3.7)** — the floor that plateaus hide below.
- **Haar measure** — the probability measure on pure states that is approximated by deep random circuits.
- **2-designs** — discrete approximation to Haar measure; deep-enough circuits form (approximate) 2-designs.

- **Vanishing gradients (classical DL)** — the similar-sounding but structurally different phenomenon in deep neural nets.

Worked Example — A 4-Qubit HEA Gradient Experiment

Consider a 4-qubit HEA with 4 layers, each layer consisting of $R_Y(\theta_{i,q})R_Z(\theta'_{i,q})$ on each qubit followed by a ladder of CNOTs. The cost is $C = \langle Z_0 \rangle$.

At $n = 4$, this has 32 parameters. Draw 1000 random parameter vectors $\boldsymbol{\theta}^{(r)}$ uniformly in $[0, 2\pi)^{32}$. For each, compute the partial derivative $\partial C/\partial\theta_1$ via the parameter-shift rule.

Empirically (reproducing McClean et al.): - Mean over 1000 samples: ≈ 0 (as expected — the mean gradient is zero by symmetry). - Variance over 1000 samples: ≈ 0.05. - Typical magnitude: ≈ 0.22.

At $n = 4$, the plateau isn't fully developed yet — the gradient is small but detectable.

Now scale up to $n = 8$ (with the same depth scaling): - Variance: ≈ 0.004. - Typical magnitude: ≈ 0.06.

At $n = 12$: - Variance: ≈ 0.0003. - Typical magnitude: ≈ 0.017.

Extrapolating, at $n = 20$ the typical gradient is $\sim 10^{-3}$, and at $n = 30$ it is $\sim 3 \times 10^{-5}$.

For comparison, the shot-noise floor at $N = 1000$ shots per evaluation is ~ 0.03. At $n = 12$ the gradient is already below the shot-noise floor. Below that qubit count, the plateau starts *operationally* — the gradient signal is buried in shot noise and no descent is possible.

The plateau kicks in long before the signal is mathematically zero. It kicks in when the signal falls below the noise floor — which happens around $n = 12$ for reasonable shot budgets.

This is the operational barren plateau threshold: not the theoretical 2^{-n} scaling, but the crossover with the $1/\sqrt{N}$ shot noise floor.

Visualization

Pending. A two-panel plot. Left panel: histogram of $\partial C/\partial\theta_1$ over 1000 random parameter samples for $n = 2, 4, 6, 8, 10$ qubits. The histograms narrow around zero as n grows. Right panel: log plot of gradient standard deviation vs n, showing exponential decay, with a horizontal line marking the shot-noise floor at a fixed shot budget. The intersection defines the operational plateau threshold. Caption: "The gradient signal decays exponentially. Somewhere, it crosses the noise floor."

Exercises

1. For a 2-qubit HEA with 1 layer and cost $C = \langle Z_0 \rangle$, compute the gradient analytically as a function of the parameters. Is the gradient bounded below by any non-negligible constant? What changes at 2 layers?
2. Estimate the shot budget required to detect a gradient of 10^{-4} with SNR 3. If your hardware runs at 10^3 shots/second per circuit, how long does a single gradient component take? How long for a full 100-parameter gradient?
3. (VQA) Design a 1-qubit "HEA" and show analytically that it does not have a barren plateau for any cost function. Where does the concentration-of-measure argument break down for $n = 1$?

Research Extensions

- **Gradient tomography in deeper circuits** — efficient protocols for measuring many gradient components without paying the full parameter-shift cost.
- **Noise-assisted gradient detection** — whether certain hardware noise profiles can, counterintuitively, lift the signal out of a plateau.
- **Classical-assisted initialization** — starting the quantum optimizer from a classical approximation to avoid the plateau regime.

Cross-links to Other Courses

- **Classical deep learning** — vanishing gradients, Xavier initialization, ResNets.
- **Concentration of Measure (Module 3, Node 3.6)** — the underlying mathematical phenomenon.
- **Module 5 (Regularization)** — techniques that operationally avoid the plateau by restricting optimizer dynamics.
- **Module 8 (Expressivity vs Trainability)** — the underlying tradeoff that makes plateaus unavoidable in certain ansatz regimes.

The Barren Plateau Theorem

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.2

This node is the **formal statement** of the barren plateau phenomenon introduced in node 4.1. It gives the theorem as originally stated by **McClean, Boixo, Smelyanskiy, Babbush, and Neven (2018)**, outlines the proof strategy, and unpacks each of the assumptions to make clear exactly when the theorem applies and when it doesn't.

The proof is an exercise in Haar-measure integration and t-design theory. The structure of the proof is important in its own right because it tells us *why* the plateau occurs, which in turn tells us *how to avoid it* (Section 4.6).

The Theorem (McClean et al., 2018)

Let $U(\boldsymbol{\theta})$ be a parameterized circuit of the form

$$U(\boldsymbol{\theta}) = U_L(\boldsymbol{\theta}_L) \cdots U_1(\boldsymbol{\theta}_1),$$

where each $U_l(\boldsymbol{\theta}_l)$ is a block of parameterized gates. Let $C(\boldsymbol{\theta}) = \langle 0|U^\dagger(\boldsymbol{\theta}) H U(\boldsymbol{\theta})|0\rangle$ be the cost. Assume:

1. **Approximate 2-design.** The distribution over unitaries induced by parameters on some "cut" of the circuit (either $U_{L:k+1}$ or $U_{k:1}$) forms an approximate 2-design on the relevant subspace.
2. **Traceless observable.** The observable H satisfies $\mathrm{Tr}(H) = 0$.

Then the gradient variance satisfies

$$\mathrm{Var}_{\boldsymbol{\theta}}\left[\frac{\partial C}{\partial \theta_i}\right] \leq \frac{2\,\mathrm{Tr}(H^2)}{(2^n+1)(2^n-1)} \sim O(2^{-2n}).$$

The gradient has **exponentially small variance in the number of qubits**. By Chebyshev's inequality, this implies the probability of observing a gradient larger than any fixed threshold is exponentially small.

What Each Assumption Means

The theorem's power — and its limits — comes from the two assumptions.

Assumption 1: approximate 2-design. A unitary 2-design is a distribution over unitaries that exactly reproduces the first two moments of the Haar distribution. Deep-enough random circuits form approximate 2-designs, which is what makes the Haar-integration proof work. - A circuit of depth $O(\mathrm{poly}(n))$ with random parameters on a connected qubit architecture forms a 2-design. - A circuit of depth $O(\log n)$ does **not** form a 2-design in general, and the theorem may not apply. - A circuit whose parameterization is structured (e.g., all gates on one qubit) does not form a 2-design across the full state space, and the theorem may not apply.

The assumption is both what makes the theorem universal (any sufficiently generic ansatz) and what makes its avoidance possible (specific, structured ansätze escape).

Assumption 2: traceless observable. $\mathrm{Tr}(H) = 0$ makes the Haar-expected value of $\langle H\rangle$ equal to zero, which simplifies the variance calculation. Any Pauli string other than the identity is traceless, so this assumption holds for nearly every observable of interest. For non-traceless H, you can subtract the mean $\mathrm{Tr}(H)I/2^n$ without changing the optimization, reducing to the traceless case.

Proof Sketch

The proof is a Haar-measure computation. Here's the structure:

Step 1: Set up the gradient as a Haar integral.

Write the circuit as $U = U_R W U_L$, where W is the gate containing θ_i and U_R, U_L are everything to the right and left. If $W = e^{-i\theta_i V/2}$ for some Hermitian V, the gradient is

$$\frac{\partial C}{\partial \theta_i} = \frac{i}{2}\langle 0|U_L^\dagger W^\dagger [V, U_R^\dagger H U_R] W U_L|0\rangle.$$

(This follows from directly differentiating and using the commutator identity for the derivative of the rotation gate.)

Step 2: Integrate over random U_L (or U_R) assumed to be a 2-design.

Treat U_L as Haar-distributed. Compute the expectation of $\partial C/\partial\theta_i$ over U_L:

$$\mathbb{E}_{U_L}\left[\frac{\partial C}{\partial \theta_i}\right] = 0.$$

The mean vanishes because V is traceless (for Pauli generators) and Haar averages of single operators evaluate to the identity times a constant.

Step 3: Compute the variance.

$$\mathrm{Var}_{U_L}\left[\frac{\partial C}{\partial \theta_i}\right] = \mathbb{E}_{U_L}\left[\left(\frac{\partial C}{\partial \theta_i}\right)^2\right].$$

This requires a second-moment Haar integral, which uses the fact that U_L is a 2-design. The second-moment Haar integral evaluates to a specific combination of traces:

$$\mathbb{E}_{U_L}\left[(\cdot)^2\right] = \frac{\mathrm{Tr}(H^2)\cdot\mathrm{Tr}(\rho V^2) - \mathrm{Tr}(H)^2\cdot\mathrm{Tr}(\rho V^2)/d - \cdots}{(d^2-1)}\cdots$$

where $d = 2^n$ is the Hilbert space dimension. Simplifying using tracelessness and the fact that $V^2 = I$ (for Pauli generators):

$$\mathrm{Var}\left[\frac{\partial C}{\partial \theta_i}\right] = O\left(\frac{\mathrm{Tr}(H^2)}{2^{2n}}\right).$$

Step 4: Conclude the exponential decay.

For $\mathrm{Tr}(H^2) = O(\mathrm{poly}(n))$ (which is typical for local Hamiltonians with polynomial number of terms), the variance scales as $O(\mathrm{poly}(n)/2^{2n})$, which is exponentially small.

The gradient standard deviation is therefore $O(\mathrm{poly}(n)/2^n)$, matching the empirical observations from 4.1.

What The Proof Really Shows

Two non-obvious implications of the proof structure:

1. The Hamiltonian matters, but only via $\mathrm{Tr}(H^2)$**.** Different Hamiltonians give different prefactors but the same 2^{-2n} scaling. You cannot "pick a better Hamiltonian" to avoid the plateau — the scaling is fixed by the dimension, not by the Hamiltonian. You *can* avoid it by picking a *structurally different* cost function (local observables, Section 4.6), but that is a cost-function redesign, not a Hamiltonian tweak.

2. The plateau is a statement about random ansätze, not specific parameter points. The theorem says: averaged over random parameters, the variance is exponentially small. It does **not** say: at every parameter point, the gradient is exponentially small. There may be specific parameter points where the gradient is large (the "good regions" of parameter space). The problem is that they are exponentially rare, so a random initialization is exponentially unlikely to land in one. This is why **warm starts** and **structured initialization** work (Section 4.6) — they avoid the random regime entirely.

These two implications are what the mitigation strategies in the rest of Module 4 exploit.

Refinements and Generalizations

Since McClean et al. 2018, the theorem has been extended in several directions.

Cerezo, Sone, Volkoff, Cincio, Coles (2021): "Cost function dependent barren plateaus." The theorem was generalized to cost functions built from local observables (acting on $O(\log n)$ qubits). The conclusion: **local cost functions admit polynomial gradient scaling in the shallow regime**. Deep circuits still concentrate, but for $O(\log n)$ depth and local costs, the gradient is $\mathrm{poly}(n)^{-1}$ rather than exponential.

Marrero, Kieferová, Wiebe (2021): "Entanglement-induced barren plateaus." Extended the theorem to entanglement-based reasoning: if the ansatz produces states with volume-law entanglement, the reduced density matrices of any subsystem are maximally mixed, and any local observable has vanishing variance — a barren plateau by a different route.

Arrasmith, Holmes, Cerezo, Coles (2022): "Equivalence of quantum barren plateaus to cost concentration and narrow gorges." Showed that barren plateaus are equivalent to cost concentration — if the gradient is exponentially small, so is the cost variance. This unifies several previously-distinct characterizations.

Ragone et al. (2024): "A Lie algebraic theory of barren plateaus for deep parameterized quantum circuits." Gave a Lie-algebra-based proof that unifies several barren plateau results. The plateau is governed by the "dynamical Lie algebra" of the generators — if that algebra is large (expressive ansatz), plateaus are severe; if small (structured ansatz), plateaus are mild.

These refinements matter because they give **positive** results (conditions under which the plateau does NOT occur) in addition to the original **negative** result. Mitigation strategies in Section 4.6 are usually built on top of one of these refinements.

What The Theorem Doesn't Say

It is worth being precise about the *scope* of the theorem, because the phrase "barren plateau" is sometimes used loosely.

The theorem does NOT say:

- "Every ansatz has a barren plateau." — False. Structured, shallow, or symmetry-restricted ansätze may not. The theorem's assumption of 2-design behavior rules them out.

- "Optimization of VQAs is impossible." — False. Small-qubit instances, problem-inspired ansätze, warm-started optimization, and local cost functions all escape.
- "The plateau exists at every point in parameter space." — False. It exists *on average*; specific points may have large gradients. They are just exponentially rare.
- "Better optimizers can overcome the plateau." — False. The plateau is a property of the function, not the optimizer.

The theorem DOES say:

- For sufficiently deep, sufficiently random ansätze, the gradient variance over random initializations is exponentially small in qubit count.
- The cost of detecting a useful gradient grows exponentially in qubit count, making naive random-initialization VQA unscalable beyond $\sim 10 - 15$ qubits for hardware-efficient ansätze.
- The escape routes must come from structure — restricted ansätze, local costs, warm starts, or shallow depth.

Structural Role

This is the **formal core** of Module 4. Everything else in the module either unpacks the theorem (nodes 4.3, 4.4, 4.5 examine different mechanisms that lead to plateau behavior) or works around it (4.6, 4.7).

Module 5 (Regularization) derives much of its motivation from this theorem: regularization is one of the ways to push the optimizer into non-random regions of parameter space where the theorem's assumptions don't hold.

Relations to Other Concepts

- **Unitary 2-designs** — the probabilistic structure the theorem relies on.
- **Haar measure integration** — the computational tool used in the proof.
- **Lévy's lemma** — the classical concentration inequality this mirrors.
- **Cerezo et al. 2021** — the local-cost refinement.
- **Ragone et al. 2024** — the Lie-algebraic unification.

Worked Example — Checking The Variance Bound On A Small Instance

Consider $n = 4$, $H = Z_0$, so $\mathrm{Tr}(H^2) = \mathrm{Tr}(I) = 16$.

The theorem bound:

$$\mathrm{Var}\left[\frac{\partial C}{\partial \theta_i}\right] \leq \frac{2 \cdot 16}{(17)(15)} = \frac{32}{255} \approx 0.125.$$

Standard deviation bound: $\sqrt{0.125} \approx 0.35$.

At $n = 4$, this is a relatively mild bound — the gradient can be meaningfully nonzero.

Now $n = 10$, $H = Z_0$, $\mathrm{Tr}(H^2) = 1024$:

$$\mathrm{Var} \leq \frac{2 \cdot 1024}{(1025)(1023)} \approx 0.002.$$

Standard deviation bound: $\sqrt{0.002} \approx 0.045$.

At $n = 10$, the gradient is bounded by ~ 0.045 — already comparable to a typical shot noise floor.

At $n = 20$, $\mathrm{Tr}(H^2) = 2^{20}$:

$$\mathrm{Var} \leq \frac{2 \cdot 2^{20}}{2^{40}} \approx 2 \cdot 10^{-6}.$$

Standard deviation bound: ~ 0.0015. Well below any reasonable shot-noise floor — the plateau is fully engaged.

Note: this is an *upper bound*. The actual variance is smaller. The empirical data from Section 4.1 matches the bound's scaling but with tighter prefactors.

Visualization

Pending. A plot showing the variance bound as a function of n for different observables: single Pauli, two-local Hamiltonian, full H of a small molecule. All curves decay exponentially, with different prefactors from $\mathrm{Tr}(H^2)$*. A horizontal line at the shot-noise floor for fixed N shows the crossover. Caption: "The bound is blind to Hamiltonian structure beyond* $\mathrm{Tr}(H^2)$ *— only the scaling matters."*

Exercises

1. Compute $\mathrm{Tr}(H^2)$ for a single Pauli string, and for the sum $H = Z_0 + Z_1 + Z_0 Z_1$. How does the ratio depend on the number of terms?
2. The variance bound uses Chebyshev. Derive a corresponding upper bound on $\mathbb{P}[|\partial C/\partial\theta_i| > \epsilon]$ as a function of n and ϵ. For $n = 20$ and $\epsilon = 0.01$, is the bound useful?
3. (VQA) Show that if the ansatz is restricted to a subspace of dimension $d' < 2^n$, the variance bound becomes $O(1/d'^2)$ instead of $O(1/2^{2n})$. How does this change the scaling for a 20-qubit chemistry Hamiltonian with particle number conservation, where $d' \approx \binom{20}{10}$?

Research Extensions

- **Sharper bounds for specific ansatz families** — replacing the 2-design assumption with tighter characterizations.
- **Average-case vs worst-case analysis** — when the bound is loose and when it is tight.
- **Haar-free proofs** — derivations that don't require 2-design assumptions, using combinatorial arguments instead.
- **Finite-depth corrections** — how the variance deviates from the 2-design bound when the circuit is not deep enough.

Cross-links to Other Courses

- **Random Matrix Theory** — moments of random unitaries.
- **Representation Theory** — Weingarten functions and Haar integration.
- **Module 3 (Concentration of Measure)** — the broader phenomenon.
- **Module 8 (Expressivity vs Trainability)** — the tradeoff the theorem expresses in quantitative form.

Entanglement and Trainability

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.3

The original barren plateau theorem (4.2) is phrased in terms of unitary 2-designs and Haar averaging. This is mathematically clean but operationally opaque — "the circuit forms a 2-design" doesn't tell you what the *states* look like or what it would take to avoid the plateau.

This node gives a **second, physically transparent view** of the plateau: barren plateaus are what you get when the ansatz produces **highly entangled** states. More specifically, when an n-qubit ansatz reaches states with **volume-law** entanglement across any bipartition, the reduced density matrix of any subsystem is close to maximally mixed, and any local observable has a vanishing expectation and vanishing gradient.

This is the **Marrero-Kieferová-Wiebe (2021)** framing: barren plateaus and volume-law entanglement are two sides of the same coin.

Entanglement Scaling Laws

To set up the argument, recall the basic scaling laws for entanglement of pure n-qubit states.

Let $|\psi\rangle$ be an n-qubit pure state, and let A be a subsystem of $n_A \leq n/2$ qubits. Compute the reduced density matrix $\rho_A = \mathrm{Tr}_{\bar{A}}|\psi\rangle\langle\psi|$. The entanglement entropy of ρ_A is

$$S(\rho_A) = -\mathrm{Tr}(\rho_A \log \rho_A).$$

Three regimes, set by how $S(\rho_A)$ depends on n_A:

- **Area law:** $S(\rho_A) = O(\text{boundary area of A})$. For a 1D system, this is $S(\rho_A) = O(1)$ — constant in subsystem size. Ground states of gapped local Hamiltonians satisfy area laws.
- **Logarithmic law:** $S(\rho_A) = O(\log n_A)$. Ground states of gapless local Hamiltonians (critical points).
- **Volume law:** $S(\rho_A) = O(n_A)$ — linear in subsystem size. Haar-random states, thermal states at high temperature, states generated by random deep circuits.

The volume-law regime is the "highly entangled" regime. It saturates the maximum possible entanglement: $S(\rho_A) \leq n_A \cdot \log 2$, with equality when $\rho_A = I/2^{n_A}$ (maximally mixed).

Volume Law Implies Plateau

Theorem (Marrero, Kieferová, Wiebe, informal): If the ansatz $U(\boldsymbol{\theta})$ produces volume-law entangled states for typical $\boldsymbol{\theta}$, then:

1. The reduced density matrix $\rho_A(\boldsymbol{\theta})$ is exponentially close to the maximally mixed state $I/2^{n_A}$.
2. Any local observable O_A supported on A has expectation value $\langle O_A\rangle = \mathrm{Tr}(O_A)/2^{n_A}$, which for traceless O_A is zero.
3. The variance of $\langle O_A\rangle$ across random $\boldsymbol{\theta}$ is exponentially small in n_A.
4. The variance of the gradient of $\langle O_A\rangle$ across random $\boldsymbol{\theta}$ is exponentially small in $n - n_A$ (the complement size).

This is the barren plateau, rediscovered from a purely state-based argument. No Haar integration needed — just entanglement scaling.

Why This Matters: The Tradeoff

This framing makes the barren plateau problem look like a **tradeoff between expressivity and trainability**:

- **High expressivity** = ability to reach many different quantum states = typically implies *generating highly entangled states* = typically volume-law = typically barren plateau.
- **Low expressivity** = restricted set of states = typically lower entanglement = typically trainable.

The more states your ansatz can prepare, the more likely it is that random initialization lands on a highly-entangled state, and the more likely the gradient is unusable.

This is the central tradeoff of ansatz design. You can't have "reaches everything" and "easy to train" simultaneously — they are in direct tension.

Module 8 (Expressivity vs Trainability) is dedicated to this tradeoff in full. The point of this node is to introduce the entanglement angle on it, which is both physically clearer and useful for designing mitigations.

Entanglement-Aware Ansatz Design

The entanglement view of plateaus directly suggests mitigation strategies that focus on *controlling* entanglement:

1. Shallow circuits. Shallow circuits cannot generate volume-law entanglement. For depth d, the entanglement of a typical reduced density matrix is $O(d)$ (each gate can add $O(1)$ entanglement). So a circuit with $d \ll n$ has $O(d)$-area-law entanglement, not volume-law, and doesn't trigger the entanglement-based plateau.

2. Sparse entanglement graphs. Limit the entanglement-generating gates (CNOTs) to a specific graph. If the graph is sparse (say, a line or a tree), entanglement between distant qubits is slow to build up, and the ansatz stays in a low-entanglement regime.

3. Locality-preserving ansätze. Design gates so that each one only acts on $O(1)$ qubits. Then in constant depth, the entanglement is area law. HEA with nearest-neighbor coupling satisfies this in the shallow regime.

4. Entanglement regularization. Add a term to the cost function that penalizes excess entanglement (e.g., $\mu S(\rho_A)$). This pushes the optimizer into the low-entanglement regime, where gradients are usable.

5. Matrix product state ansätze. Use ansätze (like MPS, TTN, PEPS) that are **guaranteed by construction** to have bounded entanglement. These ansätze cannot represent volume-law states at all, so they are trivially plateau-free in the entanglement sense.

These mitigations are all justified by the entanglement view; they would be harder to motivate from the Haar-2-design view alone.

The Dual Failure Mode: Not Enough Entanglement

There is a flip side to this story worth flagging, even though Module 4 is mostly about too much entanglement.

If the ansatz produces states with *too little* entanglement, it can't represent the ground state of many interesting Hamiltonians. For example, the ground state of a 1D critical model has logarithmic entanglement, but a product-state ansatz has zero. A product-state ansatz is trivially trainable (no plateau) but represents a much smaller set of states than is needed.

The sweet spot is: enough entanglement to represent the ground state, but not so much that the plateau engages.

For quantum chemistry, ground states typically have area-law or weak-entanglement structure, so low-entanglement ansätze can work. For condensed matter and high-energy physics, ground states may have logarithmic or volume-law structure, and the tradeoff is harder to navigate.

This is one of the principal *structural* arguments for why VQAs are more tractable for chemistry than for condensed matter.

Entanglement and Shot-Noise Sensitivity

An additional consequence of the entanglement view: **high-entanglement states make the cost estimator noisier.**

For a maximally entangled state, any local Pauli has $\langle P \rangle = 0$ and variance $1 - 0 = 1$ per shot. For a low-entanglement product-like state, many local Paulis have $\langle P \rangle$ near ± 1, giving variance near 0.

So the entanglement-induced plateau is not only a gradient problem but a *cost estimation* problem. When the ansatz is in the high-entanglement regime, the cost estimator has maximum variance, *and* the gradient is exponentially small. Both effects compound — you need exponentially more shots to detect an exponentially smaller signal, a double-exponential cost.

This is why the entanglement view is operationally sharper: it predicts not just the gradient scaling but also the shot-cost scaling, and the two combine to make the plateau regime completely uneconomical.

Connecting To The 2-Design View

The two views of barren plateaus — "Haar 2-design on parameters" and "volume-law entanglement" — are **equivalent** in the sense that they describe the same physical regime.

Specifically: - A Haar-random unitary U applied to $|0\rangle$ produces a Haar-random pure state $|\psi\rangle$. - Haar-random pure states have volume-law entanglement with overwhelming probability (Page's theorem). - So any ansatz whose parameter distribution approximates a 2-design produces volume-law entangled states with overwhelming probability. - And vice versa: any ansatz that produces volume-law entangled states for typical parameters satisfies the 2-design assumption well enough for the barren plateau theorem to apply.

The 2-design view is mathematically cleaner for proving the theorem; the entanglement view is physically cleaner for motivating mitigations. Both are correct; they are the same thing told two ways.

Structural Role

This node is the **physical interpretation** of the theorem from 4.2. It explains what barren plateaus *look like* in terms of the quantum states being prepared, and sets up the depth-scaling and mitigation discussions in 4.4 and 4.6.

It is also the node that forwards cleanest to Module 8 (Expressivity vs Trainability), which develops the entanglement-expressivity-trainability tradeoff in quantitative form.

Relations to Other Concepts

- **Page's theorem** — Haar-random states have near-maximal entanglement.
- **Area laws** — entanglement of ground states of gapped local Hamiltonians.
- **Matrix product states** — tensor network ansätze with bounded entanglement by construction.
- **Entropy of entanglement** — the von Neumann entropy of the reduced density matrix.

- **Cerezo et al. (2021)** — the local cost function refinement, which can be viewed as an entanglement-aware fix.

Worked Example — Entanglement Of A Volume-Law State

Consider $n = 10$ qubits, bipartition $A = \{0, 1, 2, 3, 4\}$ and $\bar{A} = \{5, 6, 7, 8, 9\}$.

For a Haar-random state, Page's theorem gives

$$\mathbb{E}[S(\rho_A)] \approx n_A \log 2 - \frac{2^{n_A - 1}}{2^{n - n_A}} = 5 \log 2 - \frac{16}{32} = 5 \log 2 - 0.5 \approx 2.97.$$

The maximum possible entanglement is $n_A \log 2 \approx 3.47$, so a Haar-random state is within 0.5 of maximum — essentially saturated.

The reduced density matrix is

$$\rho_A \approx \frac{I_A}{32} + O(2^{-5}).$$

So for any traceless observable O_A (say $O_A = Z_0$):

$$\langle O_A \rangle = \mathrm{Tr}(\rho_A O_A) \approx 0 + O(2^{-5}) \approx 0.03.$$

The expectation is essentially zero, with fluctuation $\sim 2^{-5}$ — which matches the barren plateau variance scaling.

At $n = 20$, the same calculation gives $\langle O_A \rangle \sim 2^{-10} \approx 10^{-3}$ — in the unmeasurable regime.

This is the entanglement-based barren plateau: **the state is so scrambled that any local measurement gives a fixed answer (the Haar average), with vanishing variation across parameter changes.**

Visualization

Pending. A bar chart: entanglement entropy $S(\rho_A)$ on the y-axis, vs ansatz type on the x-axis (product, shallow HEA, deep HEA, Haar-random). A horizontal dashed line marks the volume-law threshold. Below the threshold: gradients trainable. Above: barren plateau. Caption: "Entanglement is the operational axis of the barren plateau."

Exercises

1. For a single Bell state $|00\rangle + |11\rangle$ on 2 qubits, compute the reduced density matrix on qubit 0 and verify it is maximally mixed. What is $\langle Z_0 \rangle$?
2. For a GHZ state on 3 qubits, $(|000\rangle + |111\rangle)/\sqrt{2}$, compute the reduced density matrix on the first 2 qubits. Is it maximally mixed? What does this say about the entanglement of GHZ compared to Haar-random?
3. (VQA) Design an ansatz that provably produces states with at most logarithmic entanglement. What kinds of ground states could such an ansatz reach? What does this imply about its usefulness for chemistry vs condensed matter?

Research Extensions

- **Page-curve corrections to plateau bounds** — using the full Page distribution to sharpen the barren plateau variance calculation.
- **Entanglement-regularized training** — adding entropy penalties to the cost function to keep the optimizer in trainable regions.
- **MPS-based VQA** — replacing quantum circuits with MPS ansätze (or hybrid quantum-MPS) to guarantee bounded entanglement.
- **Entanglement monitoring during training** — using intermediate entanglement measurements to detect and exit plateau regimes.

Cross-links to Other Courses

- **Quantum Information Theory** — entanglement entropy, Page's theorem, maximally mixed states.
- **Condensed Matter Physics** — area laws, topological entanglement, ground-state entanglement.
- **Tensor Networks** — MPS, PEPS, MERA — classical ansätze with bounded entanglement.
- **Module 8 (Expressivity vs Trainability)** — the full tradeoff in quantitative form.

Depth Scaling

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.4

The barren plateau theorem (4.2) says: a sufficiently-deep ansatz approximates a 2-design, and then the gradient concentrates exponentially. The entanglement view (4.3) says: a sufficiently-deep ansatz produces volume-law entangled states, and then local observables concentrate.

Both framings invoke "sufficient depth" without saying what sufficient means. **This node pins down the depth scaling.**

The short version: for a generic HEA with nearest-neighbor coupling on n qubits, the transition from "trainable" to "barren plateau" happens around depth $L \sim O(n)$ for 1D topologies, and scales more gently (like $O(n^{1/D})$) in higher-dimensional connectivities. This means there is a **shallow regime** (depth below the transition) where plateaus are mild, and a **deep regime** (above) where they are fully engaged.

The boundary is not sharp, but the scaling of the boundary with qubit count is what governs whether "add more layers" is a viable strategy for scaling up.

Depth vs 2-Design Formation

A random local circuit becomes an approximate t-design after depth $O(n \cdot t^2)$ (Brandão, Harrow, Horodecki 2016, and tighter subsequent bounds). For $t = 2$, this gives a depth threshold $O(n)$ for 2-design formation in a 1D brick-wall circuit.

Below depth $O(n)$: the distribution over states is *not* a 2-design. It is still a structured distribution with lower entanglement and larger typical gradient. **Above depth** $O(n)$: the distribution over states is approximately a 2-design, and the barren plateau theorem kicks in.

The transition is continuous — the variance of the gradient decreases as depth increases, reaching the 2^{-2n} Haar bound asymptotically. At depth $L = n/2$, the variance might be $\sim 2^{-n}$ instead of 2^{-2n} — not as bad as full Haar, but already exponentially small.

Depth Scaling In 2D And Higher

For a 2D grid topology, entanglement spreads via the boundary of a region rather than via a 1D chain. The depth needed to entangle an $L \times L$ region across its boundary is $O(L) = O(\sqrt{n})$. More generally, for a D-dimensional lattice, the depth needed to approximate a 2-design is $O(n^{1/D})$.

- 1D: depth $\sim n$ to hit the plateau regime.
- 2D: depth $\sim \sqrt{n}$.
- 3D: depth $\sim n^{1/3}$.
- All-to-all connectivity: depth $\sim \log n$.

So higher-connectivity devices (which in 2026 means superconducting qubit grids and trapped-ion all-to-all connectivity) reach the barren plateau regime *faster* than 1D devices, for the same number of gates per layer.

This is an important point: **device connectivity interacts with barren plateau depth scaling**. A more-connected device is *less* forgiving of deep ansätze because entanglement spreads faster.

The Shallow Regime

"Shallow" in the barren plateau literature typically means depth strictly less than the 2-design threshold. For 1D HEA, this is depth $L < n$. For 2D HEA on a grid, it is $L < \sqrt{n}$. For $O(\log n)$ depth, you are well inside the shallow regime for any reasonable n.

In the shallow regime:

- Entanglement is sub-volume-law, typically area-law or logarithmic.
- Reduced density matrices have structure — they are not close to maximally mixed.
- Local observables have typical expectation values that are not concentrated at zero.
- Gradients are typically $\text{poly}(n)^{-1}$, not 2^{-n} — still decreasing with n, but much more gently.

Cerezo et al. (2021) made this precise: for **local cost functions** and **shallow ($O(\log n)$ depth) circuits**, the gradient variance is bounded below by $\Omega(\text{poly}(n)^{-1})$. This is the canonical "shallow avoids barren plateaus" result, and it is the principal structural escape route.

For **global cost functions** (observables acting on all n qubits), the same result does NOT hold — even shallow circuits can have barren plateaus for global costs. This is the Cerezo et al. insight: **the cost function's locality interacts with the circuit's depth to determine trainability**.

Matching Depth To Problem Structure

A practical rule of thumb:

If your problem has area-law entanglement structure (ground state of gapped local Hamiltonian, typical chemistry ground states), use an ansatz with depth comparable to the correlation length, which is $O(1)$ or $O(\log n)$. You stay in the shallow regime and avoid barren plateaus.

If your problem has logarithmic entanglement structure (critical 1D systems), use an ansatz with $O(\log n)$ depth. You are on the edge of the shallow regime but still polynomially trainable.

If your problem has volume-law entanglement (thermalized systems, random Hamiltonians, most condensed matter problems in 2D+), you need either: - A deep ansatz, which hits the plateau → optimization is infeasible. - A shallow ansatz that can't reach the ground state → approximation is too coarse. - A structural workaround (problem-specific ansatz that encodes the volume-law structure without random 2-designs).

The third option is where most research effort goes for hard problems. ADAPT-VQE, UCCSD, and HVA are examples of problem-specific ansätze that provide structure without (necessarily) hitting the plateau regime.

The "Layerwise Training" Workaround

One elegant response to the depth problem is **layerwise training** (Skolik et al., 2021):

1. Start with a 1-layer ansatz. Optimize.
2. Add a layer, initialized to identity (so it doesn't disturb the current state). Optimize again.
3. Repeat until you reach the desired depth.

The key insight is that **at each step, the optimizer is only training a shallow ansatz**, so it never enters the barren plateau regime. The early layers get trained in a trainable landscape. By the time deep layers are added, the early layers have already done most of the work, and the deep layers only need to fine-tune.

This avoids the plateau by **never training a deep random ansatz** — it trains a sequence of shallow ansätze, each built on top of the previous. It is an example of the broader principle that **plateaus are avoided by initialization, not by optimization**.

Depth Limits In Practice

Real hardware imposes a separate depth limit: decoherence. Each additional gate adds some error, and after depth $O(1/\text{gate error rate})$, the state is so corrupted that the computation is meaningless. For 2026-era hardware with gate errors $\sim 10^{-3}$, useful depth is $O(10^2) = O(100)$ gates.

This **hardware-imposed depth limit** is sometimes *lower* than the barren plateau threshold depth, so in practice you can't reach the barren plateau regime anyway. But as gate fidelities improve, the barren plateau regime becomes reachable, and depth-scaling arguments become practically relevant.

The interplay between hardware noise (Module 9), which limits depth from above, and barren plateaus (this module), which limit useful depth from above, is what sets the operational "sweet spot" for ansatz depth on a given device. On 2026 hardware, depth $\sim 20 - 50$ is typical — well below the plateau threshold for $n \leq 20$ qubits, and limited primarily by noise. On better hardware (gate errors $< 10^{-4}$), depth will be limited by plateaus rather than noise.

Structural Role

This node is the **quantitative** bridge between the theorem (4.2) and the mitigations (4.6). It turns "the plateau exists" into "the plateau exists beyond depth $L \sim f(n)$," which is what makes it possible to actually design around it.

It also forwards to Module 8, which develops the depth-expressivity relationship in full.

Relations to Other Concepts

- **Brandão-Harrow-Horodecki 2016** — rigorous bounds on t-design formation by random circuits.
- **Lieb-Robinson bounds** — the rate at which information spreads through a local Hamiltonian.
- **Correlation length** — the characteristic length scale of a ground state, which sets the required ansatz depth.
- **Layerwise training (Skolik et al. 2021)** — the practical training strategy that avoids deep-ansatz plateaus.
- **Cerezo et al. 2021** — the local cost + shallow depth result.

Worked Example — Depth Threshold For A 1D HEA

Consider a 1D HEA on $n = 16$ qubits with nearest-neighbor CNOTs and single-qubit Pauli rotations. The 2-design depth threshold is $L \sim n = 16$.

At **depth 4**: gradient variance ~ 0.02, entanglement $\sim 0.5 \cdot \log 2$ — clearly shallow, trainable.

At **depth 8**: gradient variance ~ 0.005, entanglement $\sim 1.0 \cdot \log 2$ — still trainable but starting to feel the plateau.

At **depth 16**: gradient variance $\sim 10^{-4}$, entanglement $\sim 2.0 \cdot \log 2$ — at the boundary; empirically starting to fail.

At **depth 32**: gradient variance $\sim 10^{-8}$, entanglement saturating volume law — fully barren plateau.

At **depth 64**: gradient variance $\sim 2^{-32} \sim 10^{-10}$ — completely hopeless.

So for 16 qubits, there is a window (depth 1-12 or so) where the ansatz is both expressive enough to reach meaningful states and shallow enough to avoid plateaus. Depth 12-16 is the transition zone. Depth 16+ is barren.

This window shrinks (as a fraction of required depth) as n grows. For very large n, the window may not contain any depth that is both expressive and trainable — which is the fundamental structural problem Module 8 will analyze in depth.

Visualization

Pending. A 3D plot: qubit count n on one axis, depth L on another, gradient variance on the third (log scale). A "ridge" of trainable region is visible at shallow depths, a plateau of barren variance at deep depths. Annotated with the 2-design threshold curve $L = n$ (or $L = \sqrt{n}$ for 2D). Caption: "The trainable region shrinks as qubits grow."

Exercises

1. For a 1D nearest-neighbor HEA with 6 qubits, at what depth does the empirical gradient variance drop below 10^{-3}? (Sketch the argument, you don't need to actually run the simulation.)
2. The Brandão-Harrow-Horodecki bound gives 2-design formation at depth $O(n)$ in 1D. What is the corresponding bound for all-to-all connectivity? How does this affect the practical depth limit for trapped-ion devices?
3. (VQA) Compare the layerwise training strategy to starting the optimizer at a small-random init for the full depth. Under what conditions is layerwise strictly better?

Research Extensions

- **Tighter depth bounds for 2-design formation** — improved results beyond Brandão-Harrow-Horodecki.
- **Adaptive depth schedules** — dynamically adjusting circuit depth during training.
- **Depth-dependent learning rates** — using the known depth scaling to set optimizer hyperparameters.
- **Warm-start depth analysis** — how much depth a warm-started ansatz can tolerate without entering the plateau.

Cross-links to Other Courses

- **Quantum Information (2-designs, random circuits)** — the formal setting for depth scaling.
- **Module 2 (Compilation)** — depth as a budgeting constraint.
- **Module 8 (Expressivity vs Trainability)** — the full qualitative/quantitative treatment of the tradeoff.
- **Module 9 (Hardware Noise)** — decoherence-imposed depth limits that may dominate the plateau-imposed limits.

Noise-Induced Barren Plateaus

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.5

Everything in nodes 4.1-4.4 treated the ansatz as an ideal unitary and the noise as purely statistical (shot noise). Under that assumption, plateaus arise from the *expressivity* of the ansatz — deep, random, 2-design-like circuits concentrate.

This node introduces a **second mechanism** for barren plateaus, one that arises from **hardware noise** rather than ansatz expressivity. **Wang, Fontana, Cerezo, Sharma, Sone, Cincio, Coles (2021): "Noise-induced barren plateaus in variational quantum algorithms."**

The short version: when every layer of an ansatz is followed by incoherent noise (e.g., depolarizing noise), the effective state after L layers is exponentially close to the maximally mixed state $I/2^n$, for *any* initial state and *any* parameters. On the maximally mixed state, every observable has the same expectation value (its trace over dimension), so the cost function is a constant — zero gradient, perfect plateau.

This is a **hardware-origin** barren plateau: the noise eats the signal at a rate set by the gate fidelity, and past a critical depth no ansatz design can recover from it.

The Noise-Induced Mechanism

Suppose every gate in the ansatz is followed by depolarizing noise with strength p:

$$\mathcal{E}_p(\rho) = (1-p)\rho + p\frac{I}{2^n}.$$

After one noisy layer, the state is a convex combination of the ideal state and the maximally mixed state. After L layers, the "ideal component" has weight $(1-p)^L$, and the "noise component" has weight $1-(1-p)^L$.

For L such that $(1-p)^L \approx 2^{-n}$, we have

$$L \approx \frac{n \log 2}{p}.$$

At this depth, the state is exponentially close to maximally mixed, **for any input and any parameters**. Every local observable has expectation value $\mathrm{Tr}(O)/2^n$, which for traceless observables is zero. The gradient is zero, and the optimization is hopeless.

The Wang et al. Theorem

Formally (informal statement):

Theorem (Wang et al. 2021): For an ansatz $U(\boldsymbol{\theta})$ interleaved with depolarizing noise channels of strength p per layer, and for any observable O with bounded norm, the gradient variance satisfies

$$\mathrm{Var}\left[\frac{\partial C}{\partial \theta_i}\right] \leq \mathrm{const} \cdot (1-p)^{2L} \cdot \|O\|^2.$$

This is **exponential in depth**, not in qubit count — a critical difference from the standard 2-design plateau.

Implications:

1. **Depth-independent from the 2-design viewpoint.** You don't need the ansatz to be deep enough to form a 2-design; noise alone creates the plateau if depth exceeds the noise-dominated threshold $L \sim 1/p$.
2. **Qubit-count-independent.** The plateau depth threshold depends on the noise rate, not the qubit count. A 4-qubit noisy deep circuit has the same kind of plateau as a 40-qubit one, if the per-gate noise rate is the same.
3. **Unavoidable by ansatz redesign.** Restricting the ansatz to a low-entanglement subspace does NOT help, because the noise acts on every state, not just on highly entangled ones. You need better *hardware* (lower p) or shallower circuits.
4. **Local cost functions don't help.** Unlike the Cerezo 2021 result, local cost functions give the same noise-induced plateau scaling. The issue is that the full state is corrupted, so no local observation helps.

This makes noise-induced plateaus a qualitatively **harder** problem than 2-design plateaus: the usual ansatz-side fixes don't work.

Practical Depth Limit From Noise

For a device with gate error rate p, the depth at which noise-induced plateaus engage is $L \sim 1/p$.

On 2026 superconducting hardware: - Single-qubit gate error: $\sim 10^{-4}$. - Two-qubit gate error: $\sim 10^{-3}$.

The effective per-layer noise rate is dominated by two-qubit gates, so $p \sim 10^{-3}$ and the plateau threshold is at depth $L \sim 10^3$. This is *far* beyond the depths achievable in current VQA experiments (typically $L < 50$), so noise-induced plateaus are not the bottleneck on today's hardware.

On older hardware with $p \sim 10^{-2}$, the threshold is at depth $L \sim 100$, which *is* reachable and was part of why early VQAs (2016-2020) struggled.

The relationship is simple: **as hardware improves, noise-induced plateaus recede**. This is different from 2-design plateaus, which get *worse* as you try to scale to more qubits. The two plateaus have opposite trends with respect to "more hardware."

Noise-Induced vs 2-Design Plateau: Comparison

Feature	2-Design Plateau	Noise-Induced Plateau
Origin	Ansatz expressivity (Haar distribution)	Hardware noise (depolarizing)
Scaling	$O(2^{-2n})$ variance, exponential in n	$O((1-p)^{2L})$ variance, exponential in L
Triggered by	Deep ansatz, random init	Any depth, bad hardware
Shallow circuits?	Can avoid	Still affected, just less
Local cost help?	Yes (Cerezo 2021)	No
Ansatz redesign help?	Yes (structure, restriction)	No
Fix	Restrict ansatz, warm start, local cost	Lower gate error, shorter depth
Trend with hardware progress	Gets worse as n grows	Gets better as p shrinks

This table is why the two mechanisms need separate treatment. They look the same operationally (flat training curves) but require different remedies.

Diagnosing Which Plateau You're In

If the training curve is flat, how do you tell which mechanism is responsible?

Test 1: Noiseless simulation. Simulate the same ansatz and cost function without hardware noise, using exact statevector. If the training still fails, it's a 2-design plateau (or some other ansatz/cost problem). If the training succeeds in simulation but fails on hardware, noise is contributing.

Test 2: Vary the depth. Reduce the circuit depth. If the plateau improves, noise is a significant factor (shorter circuits have less accumulated noise). If the plateau stays the same or gets worse as depth decreases, noise is not the dominant mechanism.

Test 3: Vary the qubit count. Reduce the qubit count at fixed depth. If the plateau improves significantly, you're in the 2-design regime (the plateau depends on n). If it stays the same, you're in the noise regime.

Test 4: Measure the effective quantum state. Tomography on the final state, or purity measurement. If the state is close to maximally mixed (purity $\sim 1/2^n$), you're in the noise regime. If the state is pure but the cost is flat, you're in the 2-design regime.

These diagnostics matter because they tell you where to invest effort: on the ansatz side, on the hardware side, or on both.

Coherent vs Incoherent Noise

The Wang et al. 2021 result assumes **incoherent** (depolarizing, dephasing) noise. **Coherent** noise — small systematic rotations of unitaries due to miscalibration — behaves differently. Coherent noise is a perturbation of the ideal unitary and doesn't directly concentrate the state. It can introduce optimization problems (shifted minima, accumulated phase errors) but doesn't create the noise-induced plateau per se.

In practice, real hardware has both: coherent errors from miscalibration (which you can mitigate via calibration or randomized compiling) and incoherent errors from decoherence (which are the dominant contributor to noise-induced plateaus). Randomized compiling (Wallman-Emerson) can convert coherent noise into incoherent noise, which makes the noise profile *simpler* but can also shift you closer to the noise-induced plateau regime.

The Noise-Plateau vs Noise-Optimization Distinction

There is a subtlety worth flagging: "noise makes optimization hard" is not the same as "noise-induced barren plateaus."

Shot noise (Node 3.7) corrupts the estimate of the cost function but doesn't touch the underlying state. More shots fixes it. It is a statistical problem.

Hardware noise (this node) corrupts the quantum state itself, not just the estimate. More shots do *not* fix it — the mean of the estimator is biased away from the true cost. It is a physical problem.

The confusion between the two has led to a few misunderstandings in the literature. When someone says "noise makes optimization slow," they might mean shot noise (fixable) or hardware noise (not fixable via statistics). These are genuinely different.

Module 9 (Hardware Noise Models) develops this distinction in more detail. For this node, the key point is: **noise-induced plateaus are a physical phenomenon where the state is wrong, not a statistical phenomenon where the estimator is noisy**.

Structural Role

This node is the **noise-origin** node of Module 4. Where 4.1-4.4 discussed plateaus as a consequence of ansatz expressivity, this node introduces plateaus as a consequence of hardware noise. It is a bridge to Module 9 (Hardware Noise Models), which treats noise in full.

Relations to Other Concepts

- **Depolarizing channel** — the canonical incoherent noise model.
- **Wang et al. 2021** — the original noise-induced barren plateau paper.
- **Randomized compiling (Wallman-Emerson)** — the technique for converting coherent to incoherent noise.
- **Error mitigation** — techniques to estimate the noise-free cost from noisy measurements.
- **Module 9 (Hardware Noise)** — the broader treatment of noise channels.

Worked Example — Depth Threshold For Noise-Induced Plateau

Consider a device with two-qubit gate error rate $p = 10^{-3}$, single-qubit error rate 10^{-4}. An HEA layer on 10 qubits has roughly 10 two-qubit gates (nearest-neighbor CNOTs), so the effective per-layer noise rate is $p_{\text{layer}} \approx 10 \cdot 10^{-3} = 10^{-2}$.

The fidelity after L layers is $(1 - p_{\text{layer}})^L \approx e^{-0.01L}$.

At $L = 10$: fidelity ≈ 0.9. Mild noise, trainable. At $L = 50$: fidelity ≈ 0.6. Significant but usable. At $L = 100$: fidelity ≈ 0.37. Borderline. At $L = 200$: fidelity ≈ 0.13. Noise is dominant. At $L = 500$: fidelity ≈ 0.007. State is essentially maximally mixed. Noise-induced plateau engaged.

For scaling to $n = 20$, the per-layer noise rate roughly doubles (20 two-qubit gates per layer), so the thresholds shift to shorter depths. **The noise-induced plateau depth decreases as n grows**, because more gates per layer means more noise per layer. This is opposite to the 2-design plateau, which requires *more* depth to reach at larger n.

So at large n, the operationally-reachable depths may be in a window bounded *above* by noise and *above* (also) by expressivity-driven plateaus. This compresses the trainable depth window further.

Visualization

Pending. A 2D plot with depth L on the x-axis and qubit count n on the y-axis. Shaded regions: "2-design plateau" (upper-right), "noise-induced plateau" (lower-right, curving), "trainable region" (lower-left). As n grows, the trainable region shrinks from both sides. Caption: "Two plateaus, two directions. The trainable window shrinks as scale grows."

Exercises

1. For a device with single-qubit error 10^{-3} and two-qubit error 10^{-2}, compute the depth at which a 10-qubit ansatz crosses into the noise-induced plateau regime (state purity $< 1/2$).
2. Show that randomized compiling preserves the expected cost function but modifies the variance of the estimator. How does this interact with noise-induced plateaus?
3. (VQA) For a fixed device with noise rate p, what is the maximum number of qubits that can be meaningfully trained before noise-induced plateaus block progress? How does this depend on the required circuit depth for the problem?

Research Extensions

- **Error-mitigated barren plateaus** — whether ZNE or CDR can extend the trainable depth.
- **Noise-aware ansatz design** — ansätze that are robust to realistic noise profiles.
- **Bias-corrected gradient estimators** — using noise models to remove the bias that noise introduces.
- **Plateau-noise phase diagrams** — precise characterization of which (n, L, p) regions are trainable.

Cross-links to Other Courses

- **Open Quantum Systems** — master equations, depolarizing channel, dephasing.
- **Error Mitigation** — zero-noise extrapolation, Clifford data regression, probabilistic error cancellation.
- **Module 9 (Hardware Noise Models)** — the full treatment of noise in VQAs.
- **Module 3 (Shot Noise)** — the *other* noise that affects optimization.

Strategies for Avoiding Barren Plateaus

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.6

This is the **practical** node of Module 4 — the one that takes everything developed so far about *why* barren plateaus happen and turns it into a catalogue of strategies for *what to do about them*.

There is no single trick that defeats barren plateaus. There is a **toolkit** of complementary strategies, each targeting a different mechanism, and a VQA practitioner's job is to match the strategy to the cause.

This node walks through the main categories, explains the assumptions behind each, and gives practical guidance on when to reach for which.

A Map Of The Strategies

The strategies fall into six broad categories:

1. **Restrict the ansatz** — make it less expressive so it can't reach volume-law entangled states.
2. **Restrict the cost function** — use local observables that have polynomial gradient scaling.
3. **Warm-start the optimizer** — initialize near a known good point, avoiding the random regime.
4. **Shallow training schedules** — train shallow first, add layers gradually (layerwise training).
5. **Symmetry-preserving design** — use the structure of the problem to shrink the effective Hilbert space.
6. **Regularize the optimizer dynamics** — use regularization (Module 5) to avoid plateau regions.

Each strategy attacks a different mechanism: - Strategies 1, 3, 4, 5 attack the *ansatz-side* cause (2-design formation). - Strategy 2 attacks the *cost-side* cause (observable locality). - Strategy 6 attacks the *dynamics-side* cause (where the optimizer goes).

The right combination depends on the problem. A difficult VQE problem typically needs 3-4 of these in combination.

Strategy 1: Restrict The Ansatz

Mechanism: Use an ansatz that is structurally prevented from forming a 2-design.

Examples:

- **Hamiltonian variational ansatz (HVA):** restrict gates to generators that come from the Hamiltonian itself. This ansatz only reaches states that are in the "reachable set" from the Hamiltonian's native dynamics, which is much smaller than the full Hilbert space.
- **UCCSD / UCCSDT:** use chemically-inspired excitations, which are strongly restricted by particle number and spin symmetries.
- **ADAPT-VQE:** grow the ansatz operator-by-operator, choosing operators based on their gradient contribution. This adaptively selects a small, structured subspace.
- **Quantum convolutional neural networks (QCNN):** use translationally-invariant, constant-depth layers, which are provably plateau-free.

Assumption: The restriction is compatible with the problem — the subspace the ansatz can reach contains the ground state (or a good approximation).

When to use: Whenever the problem has structure that can be encoded in the ansatz (symmetries, locality, known physics).

Strategy 2: Restrict The Cost Function

Mechanism: Use a cost function built from local observables, which (by the Cerezo 2021 theorem) have polynomial gradient scaling in the shallow regime.

Examples:

- **Local VQE:** replace $\langle H \rangle$ with $\frac{1}{k}\sum_j \langle O_j \rangle$ where each O_j acts on a few qubits. The minimum of the local cost is (ideally) the same as the minimum of the global cost, but the gradient is much better-behaved.
- **Variance-based cost:** $\mathrm{Var}(H) = \langle H^2 \rangle - \langle H \rangle^2$. Has zero gradient only at exact eigenstates (not at arbitrary stationary points), and its local structure often yields better gradient scaling.
- **Local fidelity cost:** for state preparation, use $\sum_j \|\rho_j - \sigma_j\|^2$ where ρ_j, σ_j are reduced density matrices.

Assumption: The local cost function has the same global minimum as the global cost function (or at least approximately so).

When to use: Whenever the global cost function can be rewritten or approximated locally. Particularly effective for state preparation and ground state finding.

Strategy 3: Warm-Start Initialization

Mechanism: Initialize the optimizer at a point that is *known* to have large gradient, rather than a random point.

Examples:

- **Hartree-Fock initialization:** for chemistry, start the ansatz at parameters that prepare the Hartree-Fock state. This is usually close to the ground state and has meaningful gradients.
- **Classically-computed warm start:** use a classical algorithm (DMRG, CCSD, MPS) to compute an approximate ground state, then reverse-engineer an ansatz that prepares it. Start the optimizer at those parameters.
- **Physics-based priors:** use known physics (e.g., an adiabatic state preparation trajectory) to set initial parameters.
- **Identity block initialization:** initialize gates so the initial state is $|0\rangle$ or some simple reference, not a random state.

Assumption: You have access to a reasonable classical approximation, or the problem has known structure that suggests an initial point.

When to use: Whenever such an approximation exists. For chemistry, this is almost always. For general condensed matter, sometimes.

Strategy 4: Layerwise And Adaptive Training Schedules

Mechanism: Never train a deep random ansatz. Instead, train a sequence of shallow ansätze, each built on top of the previous.

Examples:

- **Layerwise training (Skolik et al. 2021):** start with 1 layer, train to convergence, freeze those parameters, add a new layer initialized to identity, train again. Repeat.
- **Progressive depth growth:** similar to layerwise, but use "soft" freezing (fine-tune all parameters, but with decreasing learning rate on earlier layers).
- **ADAPT-VQE:** adaptively add operators to the ansatz only when they have meaningful gradient.
- **Curriculum learning for VQA:** train on easier instances first (smaller n, simpler Hamiltonian), use the result to initialize harder instances.

Assumption: The shallow ansatz can reach useful states, and layers added later don't disturb what was found.

When to use: Whenever you need to train a deep ansatz and can't warm-start it. Layerwise is a good default fallback.

Strategy 5: Symmetry-Preserving Ansatz Design

Mechanism: Design the ansatz to preserve the symmetries of the Hamiltonian, so the effective Hilbert space is a symmetry-restricted subspace much smaller than 2^n.

Examples:

- **Particle-number-preserving gates:** use gates that commute with $\hat{N}$ (e.g., excitation gates). The effective Hilbert space for an n-qubit, k-particle state has dimension $\binom{n}{k}$, not 2^n. The barren plateau scales with this smaller dimension.
- **Spin-preserving gates:** similarly, preserve S^2, S_z.
- **Permutation-symmetric ansätze:** for problems with permutation symmetry, use permutation-invariant gates.
- **Z2 symmetry exploitation:** after Jordan-Wigner, many Hamiltonians have Z2 symmetries that can be tapered off, reducing qubit count by 2-4.

Assumption: The symmetry is a real property of the problem (not just an artifact of the ansatz).

When to use: Whenever the problem has exact symmetries. Combined with strategy 1 (restricted ansatz), this is often the most effective approach.

Strategy 6: Regularize The Optimizer Dynamics

Mechanism: Use regularization (Module 5) to keep the optimizer in regions of parameter space where gradients are usable.

Examples:

- **Entanglement regularization:** add a term to the cost that penalizes entanglement, keeping the optimizer in the low-entanglement regime.
- **Trust regions:** only allow updates that are small enough to stay near the current (non-plateau) point.
- **Natural gradient with Fisher regularization:** use the QFIM to rescale gradients, which implicitly penalizes movement into high-curvature (plateau) regions.
- **Variance regularization:** penalize parameter configurations with high cost variance (which correlate with plateau regions).

Assumption: The regularization target (low entanglement, small QFIM, low variance) correlates with non-plateau regions.

When to use: Module 5 makes this the *thesis-specific* strategy. The thesis argues that regularization-as-dynamical-modification is a conceptually cleaner framework than ad-hoc ansatz/cost fixes, and that the "stochastic objective generator" abstraction (Module 1.8) makes explicit why dynamics-side fixes can succeed where objective-side fixes fail.

Combining Strategies

In practice, VQA problems are rarely solved by a single strategy. The typical recipe is:

1. **Start with a problem-inspired ansatz (strategy 1)** — UCCSD or HVA for chemistry, ADAPT-VQE for general.
2. **Use a local cost function (strategy 2)** — if the problem allows.
3. **Warm-start from a classical approximation (strategy 3)** — Hartree-Fock, MPS, DMRG.
4. **Train layerwise (strategy 4)** — if starting from scratch or adding layers.
5. **Preserve symmetries (strategy 5)** — tapering, particle-number preservation.
6. **Regularize the optimizer (strategy 6)** — if the training is still struggling.

Each strategy has a *separate* effect on the plateau problem; combining them can yield multiplicative improvements.

The thesis observation is that strategies 1-5 are all **objective-side** (changing what you're optimizing), while strategy 6 is **dynamics-side** (changing how you're optimizing). These have different semantics and are best thought of as complementary, not redundant.

What Doesn't Work

A few things that might seem like good ideas but don't actually help:

"Just use more shots." Node 4.1 already debunked this. Exponentially more shots to detect an exponentially smaller signal.

"Just use a better optimizer." The plateau is in the function, not the optimizer. No optimizer can extract absent signal.

"Use classical neural network techniques." Xavier initialization, batch normalization, dropout, residual connections — these are all designed for classical vanishing gradients, which have a different mechanism. Some have quantum analogues, but naive transfer doesn't work.

"Add more layers." Not only doesn't help, it actively makes things worse (deeper = more 2-design-like = stronger plateau).

"Add more qubits for scale." The plateau depth threshold scales with qubit count, so adding qubits without also mitigating the plateau makes things worse.

Being clear about what doesn't work is as important as knowing what does. It prevents wasted effort and helps newcomers avoid predictable dead ends.

Structural Role

This node is the **practical synthesis** of Module 4. Everything in 4.1-4.5 was diagnostic; this node is therapeutic. It forwards to Module 5 (where regularization is developed in full) and to Module 8 (where the expressivity-trainability tradeoff is quantified).

The thesis's argument is made most concretely here: **strategy 6 (dynamics-side regularization) deserves a seat at the same table as strategies 1-5 (objective-side fixes)**, and Module 5 is the detailed treatment of that strategy.

Relations to Other Concepts

- **Ansatz design (Module 2)** — strategies 1 and 5 live here.
- **Cost function design (Module 3.1)** — strategy 2 lives here.
- **Warm-starting and initialization schemes** — strategy 3.
- **Layerwise / curriculum training** — strategy 4.

- **Regularization (Module 5)** — strategy 6.

Worked Example — A VQE Workflow Combining Strategies

For a 16-qubit H_2O VQE calculation:

1. **Ansatz choice:** UCCSD (strategy 1) restricted to singles and doubles. Particle-number preserving by construction (strategy 5). Symmetry-tapered down to 12 qubits (strategy 5).
2. **Cost function:** Global $\langle H \rangle$ — in this problem the global cost is natural. Don't use strategy 2.
3. **Initialization:** Hartree-Fock reference state, with UCCSD parameters initialized to the classical CCSD amplitudes (strategy 3).
4. **Training schedule:** Single-shot optimization (no layerwise needed — UCCSD isn't deep), with an L-BFGS-B optimizer.
5. **Regularization:** None in first attempt.

This recipe typically gives chemical accuracy (10^{-3} Hartree) for small molecules. If it fails, you add regularization (strategy 6) as a fallback.

Compare to a naive attempt: HEA with random init, global cost, vanilla gradient descent, no regularization. That fails for $n > 10$ with certainty.

The difference between the two approaches is not one strategy — it is the whole toolkit working together.

Visualization

Pending. A Venn diagram with three circles labeled "ansatz restriction," "cost locality," "optimizer warm-start/regularization." Different VQA recipes highlighted as overlaps of 1, 2, or 3 circles. Caption: "No single strategy solves the plateau. Combine them."

Exercises

1. For a VQE problem with a 20-qubit Hamiltonian and a 30-layer HEA ansatz, rank the six strategies by expected impact. What is the minimum combination needed to make the problem tractable?
2. The Cerezo 2021 local cost theorem requires $O(\log n)$ depth. For an ansatz of depth $\log n + 1$, does the bound still apply? What if the depth is $2 \log n$?
3. (VQA) Design a workflow for training a 40-qubit quantum neural network on a classification task. Which strategies are essential? Which are nice-to-have?

Research Extensions

- **Automatic strategy selection** — algorithms that diagnose the plateau mechanism and choose strategies accordingly.
- **Strategy composition theory** — formal analysis of when combining strategies gives multiplicative vs additive benefit.
- **Learned initialization schemes** — using meta-learning to find good warm starts across problem instances.
- **Regularization for noise-induced plateaus** — whether dynamics-side strategies can help with the Wang-et-al plateau (probably not, but worth investigating).

Cross-links to Other Courses

- **Module 2 (Parameterized Circuits)** — ansatz design choices.
- **Module 3 (Cost Landscapes)** — the underlying geometry being navigated.
- **Module 5 (Regularization)** — the thesis-specific strategy 6.
- **Module 8 (Expressivity vs Trainability)** — the tradeoff that all strategies navigate.

Plateaus vs Traps

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Barren Plateaus **Node:** 4.7

This is the **closing node** of Module 4. It addresses a subtle but important confusion in the VQA literature: **not every "stuck optimizer" is a barren plateau.**

There are at least three distinct failure modes of VQA optimization that all *look* the same from a training-curve perspective — the cost stops decreasing — but have different causes and different fixes:

1. **Barren plateau** — the gradient is exponentially small because the cost function is concentrated.
2. **Narrow gorge** — the gradient is large but in a region of exponentially small volume; the optimizer never finds it.
3. **Local trap** — the optimizer is stuck at a non-global local minimum. Gradient is zero (not small), but the cost is not globally minimized.

This node distinguishes the three, explains why they require different remedies, and connects them back to the Arrasmith et al. (2022) equivalence results.

The Three Failure Modes In Detail

1. Barren Plateau

Symptom: Cost is flat across wide regions of parameter space. Gradient magnitudes below the shot-noise floor. **Cause:** Concentration of measure — the cost function is essentially constant over exponentially much of parameter space. **Topology:** Flat. **Fix:** Restrict ansatz, warm-start, use local cost, layerwise training. **Key reference:** McClean et al. 2018.

2. Narrow Gorge

Symptom: The cost function has a minimum that is deep but *narrow* — a thin valley in parameter space. Random initialization rarely lands in it, and once in it, the optimizer can easily fall out (overshoot and go back to the plateau region). **Cause:** High curvature in the minimum direction; exponentially small volume of the attractor basin. **Topology:** Thin valley in an otherwise flat landscape. **Fix:** Precise warm-starting, small step sizes near the gorge, trust-region optimization. **Key reference:** Arrasmith et al. 2022.

3. Local Trap

Symptom: The cost stops decreasing and the gradient is near zero, but the final cost is well above the global minimum. The optimizer has "converged" to a local minimum that isn't the answer. **Cause:** The cost function has multiple local minima, and the optimizer's trajectory led it to a sub-optimal one. **Topology:** Rugged landscape with multiple basins. **Fix:** Multi-start, population methods, annealing, basin-hopping. **Key reference:** standard non-convex optimization literature.

The Arrasmith Equivalence

Arrasmith, Holmes, Cerezo, and Coles (2022) proved a surprising equivalence:

Theorem (informal): In the regime where the barren plateau theorem applies (2-design ansatz, traceless observables), the following three statements are equivalent:

1. The cost function has exponentially small gradient variance (barren plateau).
2. The cost function has exponentially small cost variance (cost concentration).
3. The cost function has exponentially small gradient magnitude outside a narrow gorge.

This is a unification: the barren plateau phenomenon is not separate from narrow-gorge phenomena, and not separate from cost concentration. They are all the same underlying fact, viewed from different angles.

Practical implication: Even if your landscape isn't *literally* flat everywhere (it has a narrow gorge somewhere), the operational problem is the same — random initialization doesn't find the gorge, and the vast majority of parameter space is plateau. The optimizer behaves identically in both cases.

So while it's conceptually useful to distinguish plateaus from gorges, for most practical VQA situations they come as a package.

Local Traps Are A Different Beast

Local traps, by contrast, are a **distinct** phenomenon. In a true local trap:

- The gradient is genuinely zero (not small-but-nonzero).
- The cost is at a *local* minimum but far from the *global* minimum.
- The trap is surrounded by ascending directions in all dimensions.
- More shots don't help (there is no signal to extract — the gradient really is zero).

This is the classical non-convex optimization problem, and it is solved by classical methods:

- **Multi-start:** run many independent optimizations from different initial points.
- **Population methods:** HOPSO, CMA-ES, genetic algorithms. Maintain a population of points that collectively explore.
- **Simulated annealing / basin-hopping:** use stochasticity to escape local minima.
- **Warm-start from a known good region:** if you know where the global minimum roughly is, start there.

These are the same techniques used in classical non-convex optimization, and they apply to VQAs with the caveat that each "sample" is expensive (quantum hardware time).

How To Tell Which One You're In

Practical diagnostics:

Gradient magnitude check. - Barren plateau / narrow gorge: gradient is below the shot noise floor. If you double your shots, the gradient doesn't improve. - Local trap: gradient is exactly zero (within shot noise, but not below it — there is literally no first-order signal because you're at a critical point).

Hessian check. - Plateau: Hessian eigenvalues are all exponentially small. The landscape is flat in all directions. - Gorge: Hessian has some exponentially large eigenvalues (narrow directions) and some small ones. - Trap: Hessian has mostly positive eigenvalues, landscape is locally bowl-shaped but the bowl is at the wrong cost value.

Cost value check. - Plateau: cost is near $\mathrm{Tr}(H)/2^n$, the Haar average. - Gorge: cost is slightly below the Haar average. - Trap: cost is significantly below the Haar average but significantly above the known ground state energy.

Multi-start check. - Plateau: all starts fail identically. The cost stays near Haar average for all runs. - Gorge: most starts fail; a few succeed. The distribution of final costs is bimodal. - Trap: different starts converge to different final costs. The distribution of final costs is multi-modal.

The multi-start check is probably the cleanest diagnostic in practice — run 10 random starts and look at the spread of final costs.

Why The Distinction Matters

The key practical reason to distinguish these three failure modes is that **they call for different fixes**.

If you diagnose a barren plateau and apply trap-escaping techniques (multi-start, population methods), you waste effort — every start lands in the plateau and nothing escapes.

If you diagnose a trap and apply plateau-escaping techniques (warm-start, local cost), you might help or might not — depending on whether the warm start happens to land in a different basin.

If you diagnose a narrow gorge and apply general plateau fixes, you might succeed by warm-starting near the gorge, but you still need small step sizes to avoid falling out.

Getting the diagnosis right saves compute. The diagnosis is not always obvious from the training curve alone — you need to do the multi-start check or Hessian analysis to be sure.

The Thesis Angle

The thesis's "stochastic objective generator" framing (Module 1.8) applies here in an instructive way:

From the optimizer's point of view, it is looking at a noisy scalar oracle. The oracle's behavior — flat everywhere, thin valley, multiple basins — is a property of the ansatz-Hamiltonian-cost combination, not of the optimizer. The optimizer can react to what it sees but cannot change what is produced.

Therefore: diagnosing what the oracle is producing (plateau, gorge, trap) is the first step, and choosing a response is the second. Conflating them leads to applying wrong remedies to the wrong problem — which is a surprisingly common failure mode in VQA practice.

The thesis argues that Module 5's regularization-as-dynamics-modification is *complementary* to the objective-side fixes in Module 4.6 specifically because they address different parts of this picture: objective-side fixes change *what the oracle produces*, and dynamics-side fixes change *how the optimizer responds to what it sees*. You often need both.

Closing Module 4

With this node, Module 4 is complete. You now have:

- **Phenomenology (4.1)** — what gradient vanishing looks like.
- **Theorem (4.2)** — the formal McClean et al. result.
- **Entanglement view (4.3)** — the physical interpretation.
- **Depth scaling (4.4)** — the $O(n)$ / $O(\sqrt{n})$ / $O(\log n)$ thresholds.
- **Noise-induced (4.5)** — the hardware-origin mechanism.
- **Strategies (4.6)** — the six-category mitigation toolkit.
- **Plateaus vs traps (4.7, this node)** — the diagnostic vocabulary for distinguishing failure modes.

Module 5 will use all of this to motivate regularization as a principled approach to the dynamics side of the problem. Module 6 will use the gradient-computation framework from 3.8 and the noise framework from 3.7 + 4.5 to design optimizers that are plateau-aware. Module 8 will take the expressivity-trainability tradeoff that has been implicit throughout and develop it as the central structural concept of VQA theory. Module 9 will take 4.5 and build out the full hardware noise treatment.

Barren plateaus are the single most important structural obstacle to scalable VQAs. Understanding them is understanding the main reason VQAs are hard. With this module done, you have the full vocabulary and formal machinery to reason about this obstacle and design around it.

Structural Role

This is the **diagnostic closure** of Module 4. It sharpens the vocabulary so that a practitioner can correctly diagnose which failure mode they are in, rather than treating all training-curve failures as "barren plateaus" by default.

It also forwards directly to Module 5, which makes the dynamics-side argument explicit.

Relations to Other Concepts

- **Arrasmith et al. 2022** — the equivalence of plateaus, cost concentration, and narrow gorges.
- **Non-convex optimization** — classical theory of local traps and escape methods.
- **Population-based optimization (HOPSO, CMA-ES)** — trap-escaping techniques.
- **Trust-region methods** — gorge-navigating techniques.
- **Basin of attraction** — the classical concept formalizing "which start points lead to which minimum."

Worked Example — Three Failure Modes In A 4-Qubit System

Consider a 4-qubit HEA with 4 layers and cost $C = \langle H \rangle$ for some Hamiltonian H.

Scenario A (plateau): $H = Z_0 \otimes Z_1 \otimes Z_2 \otimes Z_3$, a global (4-local) observable. Random initialization $\rightarrow$ 100 runs $\rightarrow$ all cost values within σ_{shot} of zero. Multi-start check: unimodal at zero. **Diagnosis: barren plateau.**

Scenario B (gorge): $H = Z_0$, a single-qubit observable. Random initialization $\rightarrow$ 100 runs $\rightarrow$ most converge to cost ~ 0, but a few ($\sim$5%) converge to cost -1 (the ground state). Multi-start check: bimodal. **Diagnosis: narrow gorge + plateau.** The narrow-gorge successful runs are rare but visible; a warm start near any of them would succeed reliably.

Scenario C (trap): $H = -X_0Y_1Z_2 - Y_0Z_1X_2 + Z_0Y_1X_2$, a designed rugged Hamiltonian with multiple local minima at distinct energies. Random initialization $\rightarrow$ 100 runs $\rightarrow$ final costs cluster at several discrete values: $-1.5, -1.0, -0.5, 0.0$. Multi-start check: multi-modal. **Diagnosis: local traps.** The optimizer needs population methods or multi-start to find the global minimum.

These scenarios illustrate that the same *ansatz* and *optimizer* can fall into different failure modes depending on the Hamiltonian. The distinction is not a property of the optimizer — it's a property of the cost landscape.

Visualization

Pending. A 3-panel cartoon of 1D cost functions: (left) flat line with tiny wiggles = plateau; (middle) flat line with one deep narrow notch = gorge; (right) rugged landscape with multiple minima at different heights = trap. Each annotated with its characteristic training curve and recommended fix. Caption: "Three failure modes, three fixes. Diagnose before treating."

Exercises

1. For a 2-qubit cost function $C(\theta_1, \theta_2) = \cos(10\theta_1)\cos(10\theta_2)$, characterize the failure mode. Is it a plateau, a gorge, a trap, or some combination? What would a multi-start test reveal?
2. Design a small (3-4 qubit) Hamiltonian and ansatz combination that exhibits a narrow gorge clearly. What parameters make the gorge narrower?
3. (VQA) Given a training run that shows flat cost curve, design a diagnostic protocol using 3-5 follow-up experiments that determines which failure mode is responsible. Estimate the total shot budget.

Research Extensions

- **Automatic failure-mode diagnosis** — algorithms that analyze training curves and diagnose which failure mode is present.
- **Sharp gorge-width bounds** — quantifying how narrow the gorges in plateau regimes typically are.
- **Hybrid plateau/trap remediation** — methods that work simultaneously on both issues.
- **Plateau-trap phase diagrams** — characterizing which ansatz-cost combinations produce which failure modes.

Cross-links to Other Courses

- **Non-convex Optimization** — classical theory of local minima, trust regions, basin-hopping.
- **Statistical Mechanics of Disordered Systems** — spin glasses as a natural model of rugged landscapes.
- **Module 5 (Regularization)** — dynamics-side responses to all three failure modes.
- **Module 6 (Optimization Dynamics)** — optimizer design with plateau/trap awareness.

Module 5 — Regularization

Regularization as Prior Injection

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.1

This is the **standard machine-learning lens** on what regularization is. It is the lens you will encounter in any textbook on supervised learning, statistics, or inverse problems. We start here because Module 5 must first explain regularization in its conventional form before we can argue (in node 5.5) that the conventional form is incomplete in the VQA setting.

Formal Definition

Given an optimization problem

$$\min_{\boldsymbol{\theta}} \; C(\boldsymbol{\theta})$$

a **regularizer** is an auxiliary penalty function $R(\boldsymbol{\theta})$ added to the objective:

$$\min_{\boldsymbol{\theta}} \; C(\boldsymbol{\theta}) + \lambda \, R(\boldsymbol{\theta}),$$

where $\lambda \geq 0$ is the **regularization strength**.

The Bayesian interpretation is that this is a **maximum a posteriori (MAP)** estimate under a prior $p(\boldsymbol{\theta}) \propto \exp(-\lambda R(\boldsymbol{\theta}))$:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} \; p(\text{data} \mid \boldsymbol{\theta}) \, p(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \; \big[-\log p(\text{data} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) \big].$$

The negative log-likelihood is the cost C. The negative log-prior is λR.

Intuition

A regularizer is a way of saying:

> Among all parameter values that fit the data equally well, prefer the ones that are "small," "smooth," "sparse," or otherwise structured.

The prior is your belief about what counts as a "reasonable" $\boldsymbol{\theta}$ before you see any data.

The data tugs the optimizer in one direction. The prior tugs it in another. The MAP estimate is the equilibrium.

In the limit $\lambda \to 0$, the prior is irrelevant and you recover the maximum-likelihood estimate. In the limit $\lambda \to \infty$, the data is irrelevant and you recover the prior mode. The art is choosing λ so that the two are in productive tension.

Structural Role

This node is the **anchor** for the standard regularization techniques in nodes 5.2 (geometric), 5.3 (spectral), and 5.4 (generalization bounds). All three are concrete instances of the prior-injection framework.

It is also the **foil** for node 5.5 (regularization as dynamical modification). The thesis claim is that prior-injection is the wrong framing for VQA, because in VQA the optimizer never reaches the MAP solution. It lives in a stochastic transient regime where dynamics, not steady states, determine outcomes. You cannot understand 5.5 without first having 5.1 in hand.

Relations to Other Concepts

- **L2 / weight decay** — Gaussian prior on parameters. $R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$.
- **L1 / Lasso** — Laplace prior. $R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$. Promotes sparsity.
- **Tikhonov regularization** — the inverse-problems name for L2.
- **Ridge regression** — linear regression with L2.
- **Dropout** — implicit regularization via stochastic pruning of computation paths.

Worked Derivations

L2 regularization is a Gaussian prior

Let $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \sigma^2 I)$. Then

$$-\log p(\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\|\boldsymbol{\theta}\|_2^2 + \text{const.}$$

Setting $\lambda = 1/(2\sigma^2)$ gives the standard L2 penalty. Smaller prior variance is equivalent to stronger regularization.

MAP versus full posterior

The MAP estimate is a single point, not a distribution. It throws away all information about the posterior's spread. Bayesian methods that integrate over the full posterior (variational inference, MCMC) recover this information at higher computational cost. For VQA, even MAP is hard — see node 5.5 for why we don't actually reach MAP in practice.

Visualization

Pending. Diagram: 2D parameter space with the data-likelihood contour ellipse, the prior contour circle (centered at origin), and the MAP point at the intersection of the two pulls.

Exercises

1. Show that L2 regularization with strength λ on a linear regression problem shrinks each least-squares coefficient by a factor of $\sigma_i^2/(\sigma_i^2 + \lambda)$, where σ_i is the i-th singular value of the design matrix.
2. Derive the MAP estimate for a Bernoulli likelihood with a Beta prior. What does the regularization strength correspond to in terms of "effective prior counts"?
3. (VQA) Consider a parameterized circuit with parameters $\boldsymbol{\theta} \in [-\pi, \pi]^d$. Why does an L2 penalty have a less natural interpretation here than in unconstrained Euclidean parameter spaces?

Research Extensions

- **Empirical Bayes** — learning the prior hyperparameters from the data itself.
- **Hierarchical priors** — putting priors on the regularization strength λ.
- **Implicit regularization** — gradient descent itself acts as a regularizer in overparameterized models, even without an explicit penalty term. This is the bridge to node 5.5.

Cross-links to Other Courses

- **Statistics Module 6** — Bayesian Inference (priors, posteriors, MAP).
- **Statistics Module 10** — Model Selection (penalized likelihood, AIC/BIC, cross-validation as implicit regularization).
- **Math-Intuition** — Lagrangian formulation of constrained optimization as regularization.

Geometric Regularization

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.2

This node specializes the prior-injection framework of node 5.1 to the case where the prior is **geometric** — that is, defined on the manifold of reachable quantum states rather than on the raw parameter vector.

Formal Definition

Let $\mathcal{M} = \{|\psi(\boldsymbol{\theta})\rangle : \boldsymbol{\theta} \in \mathbb{R}^d\}$ be the ansatz manifold (see node 2.9).

A **geometric regularizer** is a penalty function

$$R_{\text{geom}}(\boldsymbol{\theta}) = d_{\mathcal{M}}\big(|\psi(\boldsymbol{\theta})\rangle, |\psi_0\rangle\big)^2$$

where $d_{\mathcal{M}}$ is a distance function on $\mathcal{M}$ and $|\psi_0\rangle$ is a reference state (often the previous iterate).

The most common choice is the **Fubini-Study metric**:

$$d_{\text{FS}}(|\psi\rangle, |\phi\rangle) = \arccos|\langle\psi|\phi\rangle|.$$

The regularized cost becomes

$$C_{\text{reg}}(\boldsymbol{\theta}) = C(\boldsymbol{\theta}) + \lambda\, d_{\text{FS}}(|\psi(\boldsymbol{\theta})\rangle, |\psi(\boldsymbol{\theta}_k)\rangle)^2,$$

evaluated at iteration k.

Intuition

L2 regularization on $\boldsymbol{\theta}$ penalizes large parameter values. But two parameter vectors that differ a lot in $\boldsymbol{\theta}$-space may produce nearly identical quantum states — and conversely. **The geometry of the parameter manifold and the geometry of the state manifold do not agree.**

Geometric regularization fixes this by penalizing motion in *state space*, which is the physically meaningful place.

This is the same idea that motivates **quantum natural gradient** (node 6.2) — both replace the Euclidean parameter metric with the Fubini-Study metric on the state manifold.

Structural Role

Geometric regularization is the **manifold-aware** member of the standard regularization family. It connects Module 5 to Module 2 (the ansatz manifold) and Module 6 (natural gradient).

It is also a stepping stone toward node 5.5: once you accept that the parameter metric is the wrong unit of "distance," you are halfway to seeing regularization as something the optimizer dynamics need to consume — not a property of an idealized objective.

Relations to Other Concepts

- **Quantum Fisher Information** (node 6.2) — the metric tensor whose square root is the local Fubini-Study distance.
- **Trust regions** (node 5.6) — bounds the step size in some metric. When the metric is FS, this is geometric regularization in disguise.
- **Mirror descent** — generalized gradient methods that respect a non-Euclidean parameter geometry.

Worked Derivations

Fubini-Study to leading order

For an infinitesimal step $\delta\boldsymbol{\theta}$ from $\boldsymbol{\theta}_k$, the Fubini-Study distance squared is

$$d_{\mathrm{FS}}^2 = \delta\boldsymbol{\theta}^\top F(\boldsymbol{\theta}_k)\,\delta\boldsymbol{\theta} + O(\|\delta\boldsymbol{\theta}\|^4),$$

where F is the quantum Fisher information matrix. At first order in $\delta\boldsymbol{\theta}$, geometric regularization is equivalent to a **state-space Tikhonov penalty** with the QFI matrix as the metric.

Equivalence to natural gradient damping

A geometrically regularized step minimizes

$$C(\boldsymbol{\theta}_k + \delta\boldsymbol{\theta}) + \lambda\,\delta\boldsymbol{\theta}^\top F(\boldsymbol{\theta}_k)\,\delta\boldsymbol{\theta}.$$

Setting the gradient to zero gives

$$\delta\boldsymbol{\theta} = -(2\lambda F + H)^{-1}\nabla C,$$

where H is the Hessian of C. For small H, this approaches the **damped natural gradient step** $-(2\lambda F)^{-1}\nabla C$.

Visualization

Pending. Diagram: parameter space (Euclidean grid) on the left, state manifold (curved surface) on the right, with the same parameter step producing different state distances depending on the local curvature.

Exercises

1. For a single-qubit ansatz $|\psi(\theta)\rangle = R_y(\theta)|0\rangle$, compute the Fubini-Study distance between $\theta = 0$ and $\theta = \pi$. Compare to the Euclidean distance $|\pi - 0| = \pi$.
2. Show that geometric regularization with the FS metric is invariant under reparameterization of the ansatz, while L2 is not.
3. Implement geometric regularization for a 2-qubit hardware-efficient ansatz and compare convergence to plain L2 on a small VQE instance.

Research Extensions

- **Information-geometric optimization** — full theory in Amari's framework.
- **Riemannian VQE** — treating the optimization as a Riemannian descent on the state manifold from the start.
- **Geodesic regularization** — penalizing total geodesic length traversed, not just final distance.

Cross-links to Other Courses

- **Statistics Module 5** — Information Geometry (Fisher information as a Riemannian metric).
- **Linear Algebra** — Riemannian metrics on submanifolds.

Spectral Regularization of Parameterized Circuits

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.3

The third standard regularization technique. Penalizes large spectral norms of intermediate operators in the parameterized circuit, controlling Lipschitz behavior of the cost as a function of $\boldsymbol{\theta}$.

Formal Definition

Let $U(\boldsymbol{\theta}) = U_L(\theta_L)\cdots U_2(\theta_2)U_1(\theta_1)$ be a parameterized unitary built from L layers.

The **spectral norm** of an operator A is

$$\|A\|_{\text{op}} = \sup_{|\psi\rangle \neq 0} \frac{\|A|\psi\rangle\|}{\||\psi\rangle\|} = \sigma_{\max}(A).$$

For a unitary, $\|U\|_{\text{op}} = 1$ trivially, so the interesting object is the spectral norm of **derivatives**:

$$R_{\text{spec}}(\boldsymbol{\theta}) = \sum_{k=1}^{L} \left\| \frac{\partial U}{\partial \theta_k} \right\|_{\text{op}}^2 .$$

Penalizing this quantity bounds how much a small change in any single parameter can perturb the output state.

Intuition

A circuit with high spectral sensitivity is one where small parameter wiggles cause large state changes. That sounds good for expressivity — but it also means large gradient *variance* under shot noise, and large *condition number* of the optimization problem. Spectral regularization is the trade: pay a small cost in expressivity to gain stability.

It is the **Lipschitz constant lens** on regularization: bound the worst-case rate of change.

Structural Role

Spectral regularization is the most "operator-theoretic" of the standard regularizers. It is the lens that talks about *what the circuit does*, not what its parameters look like. This makes it the natural bridge between regularization theory (Module 5) and expressivity theory (Module 8).

Relations to Other Concepts

- **Lipschitz constant** — $|C(\boldsymbol{\theta}) - C(\boldsymbol{\theta}')| \leq L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$. Spectral regularization bounds L.
- **Trainability bounds** (node 8.4) — gradient variance scales with spectral quantities.
- **Dynamical Lie algebra** — the algebra generated by the circuit's generators determines its spectral richness.

Worked Derivations

Spectral norm of a Pauli rotation derivative

For $U_k(\theta_k) = e^{-i\theta_k P_k/2}$ with P_k a Pauli string,

$$\frac{\partial U_k}{\partial \theta_k} = -\frac{i}{2} P_k \, U_k.$$

Since P_k and U_k are both unitary,

$$\left\| \frac{\partial U_k}{\partial \theta_k} \right\|_{\text{op}} = \frac{1}{2}.$$

So for a depth-L circuit of Pauli rotations, $R_{\text{spec}} = L/4$, independent of $\boldsymbol{\theta}$. This is why spectral regularization is more interesting for non-Pauli generators or for compound operators.

Lipschitz bound on the cost

If $R_{\text{spec}}(\boldsymbol{\theta}) \le B$, then for any Hermitian H with $\|H\|_{\text{op}} \le M$,

$$|C(\boldsymbol{\theta}) - C(\boldsymbol{\theta}')| \le 2M\sqrt{B}\, \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.$$

This gives an *a priori* bound on the Lipschitz constant of C, which feeds directly into the convergence theory of node 6.5.

Visualization

Pending. Diagram: same circuit drawn twice — once with low spectral norm derivatives (smooth response), once with high (jagged response) — and the corresponding cost-vs-parameter curves.

Exercises

1. Show that for a single layer $U(\theta) = e^{-i\theta H}$ with H Hermitian, the spectral regularizer reduces to $\|H\|_{\text{op}}^2$.
2. Compute R_{spec} for a 2-qubit hardware-efficient ansatz with depth 3.
3. Argue that spectral regularization can never *increase* the location of the optimum, only restrict the trajectory toward it.

Research Extensions

- **Lipschitz quantum machine learning** — formal generalization theory for VQA based on Lipschitz constants.
- **Spectral pruning** — using the spectrum of the Hessian (rather than derivatives of U) as the regularization signal.
- **Operator-norm penalties on the cost itself** — closely related to the cost-shaping techniques in node 5.7.

Cross-links to Other Courses

- **Linear Algebra** — operator norms, singular value decomposition.
- **Math-Intuition** — Lipschitz functions and continuous mappings.

Regularization and Generalization Bounds

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.4

The fourth standard regularization technique. Asks: *given* the optimizer succeeds in reducing the training cost, what does the regularizer guarantee about the cost on **unseen** problem instances?

For VQE specifically, "unseen instances" means: different shots, different noise realizations, or different molecular geometries.

Formal Definition

Let $\mathcal{D}$ be a distribution over problem instances and let $\hat{C}(\boldsymbol{\theta})$ be the empirical cost on a finite training set. A **generalization bound** is an inequality of the form

$$\mathbb{E}_{x\sim\mathcal{D}}[C(\boldsymbol{\theta}; x)] \leq \hat{C}(\boldsymbol{\theta}) + \mathcal{B}(\mathcal{H}, n, \delta)$$

with probability at least $1-\delta$ over training set draws, where $\mathcal{B}$ is a complexity term that depends on the hypothesis class $\mathcal{H}$ (the ansatz family) and the training-set size n.

A **regularizer reduces** $\mathcal{B}$ by restricting the effective complexity of $\mathcal{H}$.

Intuition

Generalization is the answer to: "My circuit got the right answer on the data I trained it on. Will it get the right answer on new data?"

Without regularization, an overparameterized ansatz can perfectly fit any training set — including the noise — and fail catastrophically on new instances. A regularizer says: "Don't fit the noise. Restrict yourself to a smaller class of functions, where overfitting is impossible."

The bound $\mathcal{B}$ is the price you pay in worst-case error for the freedom to choose any $\boldsymbol{\theta}$ in the unrestricted class. Regularization shrinks the class, which shrinks the price.

Structural Role

This is the most theoretical of the four standard regularization nodes. It is the lens that lets you state, with formal probability guarantees, *why* L2 / spectral / geometric regularization should work.

But it is also the lens that **fails most cleanly in the VQA setting**, which is why it sets up the polemic in node 5.5. The standard generalization story assumes (a) you have a well-defined data distribution, (b) you reach a fixed steady-state solution, (c) your optimizer is essentially noise-free. None of these hold for VQA. See 5.5 for the full critique.

Relations to Other Concepts

- **Rademacher complexity** — the complexity term $\mathcal{B}$ for many concrete hypothesis classes.
- **PAC-Bayes** — generalization bounds that incorporate a Bayesian prior over $\boldsymbol{\theta}$, making the connection to node 5.1 explicit.

- **Stability-based bounds** — generalization through algorithmic stability rather than hypothesis-class size.
- **VC dimension** — the combinatorial measure of capacity for binary classifiers.

Worked Derivations

A schematic Rademacher bound

For a hypothesis class $\mathcal{H}$ with Rademacher complexity $\mathfrak{R}_n(\mathcal{H})$ on samples of size n, the standard bound gives

$$\mathbb{E}[C] \leq \hat{C} + 2\,\mathfrak{R}_n(\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2n}}$$

with probability $1 - \delta$.

Spectral regularization (node 5.3) bounds $\|U(\boldsymbol{\theta})\|_{\text{Lip}}$, which in turn bounds $\mathfrak{R}_n(\mathcal{H})$, tightening the inequality.

What goes wrong for VQA

The bound assumes $\hat{C}$ is the *minimum* of the empirical objective. In VQA, $\hat{C}$ is whatever the optimizer happened to converge to — typically a stochastic transient, not a minimum. The bound still holds in principle, but it ceases to be informative because the optimizer's actual behavior is governed by *dynamics*, not by the geometry of the loss surface. This is the gap that node 5.5 fills.

Visualization

Pending. Diagram: the bound as an envelope around the empirical-cost curve, shrinking as a regularizer is applied.

Exercises

1. Compute the Rademacher complexity of a linear classifier with bounded L2 weight norm.
2. Sketch why a depth-L hardware-efficient ansatz has Rademacher complexity at least exponential in L without regularization.
3. (Conceptual) Argue why "training cost" and "test cost" are not the right concepts in VQE — and what should replace them.

Research Extensions

- **Quantum PAC-Bayes** — recent work extending PAC-Bayes bounds to variational quantum models.
- **Implicit-bias generalization** — explaining generalization without explicit regularization, by analyzing the bias of the optimizer itself. This is a direct neighbor of node 5.5.
- **Distribution-free bounds** — generalization that does not depend on a fixed training distribution.

Cross-links to Other Courses

- **Statistics Module 10** — Model Selection (cross-validation, AIC/BIC, structural risk minimization).
- **Statistics Module 7** — Asymptotic Theory (uniform convergence, empirical processes).

Regularization as Dynamical Modification (and Why It Is Not Bias)

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.5

This is the **central claim** of the regularization thesis. It is the node that motivates why Module 5 is labelled "an original contribution" rather than "standard background." Everything in nodes 5.1 through 5.4 was standard ML / inverse-problems theory. This node argues that the standard theory **misframes what regularization is doing in the VQA setting**, and proposes a replacement framing.

The thesis claim

> In variational quantum algorithms, regularization should be understood not as injecting a prior into the objective, but as **modifying the dynamics of the optimizer** under stochastic feedback. The objective remains unchanged. Only the trajectory through parameter space is altered.

Source: thesis Chapter 6.2 [@thesis-ch6].

Formal Definition

Let the unregularized parameter update rule be the discrete-time dynamical system

$$\boldsymbol{\theta}_{k+1} = \mathcal{A}\big(\boldsymbol{\theta}_k,\, \hat{C}(\boldsymbol{\theta}_k),\, \mathcal{H}_k\big),$$

where $\mathcal{A}$ is the classical optimizer, $\hat{C}$ is the noisy cost estimate, and $\mathcal{H}_k$ is the optimizer's internal state at iteration k.

A **regularizer** is a modification of this map:

$$\boldsymbol{\theta}_{k+1} = \mathcal{A}_{\text{reg}}\big(\boldsymbol{\theta}_k,\, \hat{C}(\boldsymbol{\theta}_k),\, \mathcal{H}_k;\, \lambda\big),$$

where λ controls the strength.

Crucially, $\mathcal{A}_{\text{reg}}$ is not derived from a modified cost. It is a direct modification of the update rule itself — adding a damping term, smoothing the input signal, bounding the step size, etc. The fixed points of $\mathcal{A}_{\text{reg}}$ are *not* generally those of $\arg\min C + \lambda R$.

Intuition

In a noise-free convex setting, gradient descent on $C(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$ and gradient descent on C with a damping term added to the update rule are mathematically the same thing in the limit $k \to \infty$. **They converge to the same point.** This is why the standard ML literature can use prior-injection and dynamical-modification interchangeably — at convergence, they agree.

In VQA, you never reach convergence. Shot noise, NISQ-era circuit depth limits, and finite measurement budgets mean the optimizer lives in a permanent transient regime. **The two framings disagree about what is happening in the transient.**

The prior-injection framing says: "we are searching for the regularized minimum." The dynamical-modification framing says: "we are flowing along a stochastic trajectory whose distribution at iteration k is what we care about."

The latter is the right description for VQA. The former is a useful classical limit, but it does not predict what an actual VQA run looks like at iteration 50 with 1000 shots per evaluation.

Why it is not bias

A common objection: "if you modify the dynamics, surely you are introducing bias?"

No. Or rather: not in the sense the objector means.

A *biased estimator* is one whose expected steady-state value is not the true minimum. A *dynamical modification* is one whose **trajectory** is different but whose **steady-state** (in the limit $k \to \infty$ and $\lambda \to 0$) is the same as the unmodified system.

Schematically:

$$\lim_{k\to\infty} \lim_{\lambda\to 0} \mathbb{E}[\boldsymbol{\theta}_k^{(\lambda)}] = \boldsymbol{\theta}^\star$$

where $\boldsymbol{\theta}^\star$ is the true minimizer of C.

The order of limits matters. You have to take $\lambda \to 0$ **before** $k \to \infty$ for the unbiased property to hold. This is the regime in which **the cosine schedule (node 5.8) operates**: λ is annealed to zero as k grows, ensuring that the regularizer fades out before the steady state is reached.

This is the polemic point. A static regularizer (constant λ) does introduce bias — it shifts the steady state. A scheduled regularizer that vanishes at convergence does not introduce bias — it only shapes the path. The thesis argument is that **only the scheduled, dynamical interpretation makes sense in VQA**, because static regularization in a transient regime is just adding bias for no asymptotic benefit.

Structural Role

This is the load-bearing node of Module 5. Nodes 5.6, 5.7, 5.8 are all instances of dynamical modification. Node 5.9 is the empirical evidence that the framing pays off.

It is also the link between Module 5 and Module 6 (Optimization Dynamics). The dynamical-modification framing only makes sense if you have already accepted that the optimizer is a dynamical system and not a function evaluator — see node 6.1 for the foundation.

Relations to Other Concepts

- **Stochastic approximation theory** (Robbins-Monro) — the closest classical analogue. Annealed step sizes are conceptually identical to scheduled regularization.
- **Implicit bias of SGD** — the observation that gradient descent itself produces solutions that are "regularized" without any explicit penalty, depending on the optimizer.
- **Dynamical systems / control theory** — the natural language for describing the modified update map.
- **Effective potential** — viewing $\mathcal{A}_{\text{reg}}$ as gradient flow on a modified energy landscape that depends on k and λ.

Worked Derivations

Damped gradient descent as dynamical modification

Consider gradient descent with a damping term:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla C(\boldsymbol{\theta}_k) - \eta \lambda_k \boldsymbol{\theta}_k.$$

If λ_k is constant, this is L2-regularized gradient descent and converges to $\arg\min C + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$ — a biased steady state.

If $\lambda_k \to 0$ such that $\sum_k \lambda_k = \infty$ but $\sum_k \lambda_k \eta_k < \infty$, the steady state is the unbiased $\arg\min C$. The damping shapes the path but vanishes asymptotically.

Why constant λ is bad in VQA

For constant λ, the steady state of the regularized dynamics is

$$\boldsymbol{\theta}^\star_\lambda \neq \arg\min C.$$

The displacement $\|\boldsymbol{\theta}^\star_\lambda - \arg\min C\|$ is bounded by something proportional to λ/μ, where μ is the local strong-convexity constant. In VQA, μ is small (weakly identified parameters) and λ is non-trivial (you need it for stability), so the displacement is *not negligible*. You are paying a permanent bias for a temporary stability gain. **This is the core argument for scheduled regularization.**

Visualization

Pending. Diagram: two trajectory plots in 2D parameter space — one with constant λ (ends at biased point), one with scheduled λ (ends at the true minimum, but visits a more stable transient region).

Exercises

1. Construct a 1D quadratic example where constant L2 regularization shifts the minimum by an amount you can compute exactly.
2. Show that for the same example, a cosine-scheduled L2 with $\lambda_k = \lambda_0 \cos^2(\pi k/2K)$ reaches the true minimum at iteration K.
3. Argue (without proof) why the dynamical-modification framing is unnecessary in noise-free convex optimization, and necessary in noisy non-convex settings.

Research Extensions

- **Stochastic differential equation models of SGD** — continuous-time limit of the discrete dynamics; turns regularization into a drift term.
- **Mean-field theory of optimizer dynamics** — averaging over noise realizations to predict trajectory distributions.
- **Connection to quantum annealing schedules** — the cosine schedule in 5.8 is closely related to adiabatic quantum schedules.

Cross-links to Other Courses

- **Statistics Module 9** — Stochastic Optimization (Robbins-Monro, annealed step sizes).
- **Math-Intuition** — Dynamical systems, fixed points, and basins of attraction.
- **Quantum Foundations** — Adiabatic theorem (analogous schedule structure).

Parameter-Space Regularization Techniques

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.6

This node catalogs the **parameter-space** family of regularization techniques: methods that act directly on the evolution of $\boldsymbol{\theta}$ by adding restoring forces, bounds, or step-size limits. It is one of the two operational halves of the dynamical-modification view introduced in node 5.5. Its sibling is node 5.7, which catalogs **objective-space** techniques.

The Common Mechanism

Every parameter-space technique modifies the update rule

$$\boldsymbol{\theta}_{k+1} = \mathcal{A}\big(\boldsymbol{\theta}_k, C(\boldsymbol{\theta}_k), \mathcal{H}_k\big)$$

into

$$\boldsymbol{\theta}_{k+1} = \mathcal{A}_{\text{reg}}\big(\boldsymbol{\theta}_k, C(\boldsymbol{\theta}_k), \mathcal{H}_k; \lambda\big),$$

without changing C. The intervention happens on the **dynamics side**, not the **objective side**. This is the key contrast with node 5.7.

Technique 1 — Quadratic Penalty (L2 / Weight Decay)

Adds a penalty proportional to $\|\boldsymbol{\theta}\|_2^2$ inside the optimizer's update. For gradient methods this becomes a restoring force toward the origin:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\big[\nabla C(\boldsymbol{\theta}_k) + \lambda \boldsymbol{\theta}_k\big].$$

In a periodic parameter space — which is what VQA actually has, since rotation angles live on a torus — the quadratic penalty has no natural origin and is conceptually awkward. This is why parameter-space regularization in VQA is usually applied as a **schedule** that vanishes near convergence (see node 5.8) rather than as a fixed prior.

Technique 2 — Trust Regions

Rather than penalizing parameter values, **bound the step**:

$$\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\| \leq \Delta_k.$$

Trust regions are particularly natural for VQA because objective evaluations far from the current iterate are unreliable — the stochastic estimate of C is only locally meaningful. A trust region encodes the statement: *do not trust the optimizer's predicted descent direction beyond the radius where the cost estimate is statistically valid.*

Technique 3 — Parameter Clipping

Hard bounds on parameter values:

$$\theta_i \leftarrow \max(\theta_{\min}, \min(\theta_{\max}, \theta_i)).$$

Simple but introduces non-smooth behavior at the boundaries. In VQA, clipping is most often used as a safety net against runaway behavior, not as a primary regularizer. The natural domain $\theta_i \in [-\pi, \pi]$ for rotation angles makes clipping degenerate with the periodic structure of the manifold.

Technique 4 — Step-Size Control and Velocity Damping

Adaptive learning rates, momentum decay, and (in population-based methods) velocity caps all act as parameter-space regularizers. For HOPSO specifically, the harmonic damping term

$$\boldsymbol{v}_{k+1} = \alpha \boldsymbol{v}_k + \omega^2(\boldsymbol{\theta}_k - \boldsymbol{\theta}_k^\star)$$

is itself a parameter-space stabilizer baked into the dynamics. This is why HOPSO needs less *added* regularization than naive optimizers — the structure already does some of the work (see Module 6 for the full HOPSO treatment).

Structural Role

This node enumerates the **levers** that node 5.5 abstractly named "modifications to the dynamical map." Together with node 5.7 (objective-space techniques), it provides the operational vocabulary that node 5.8 (cosine schedule and two-stage protocol) and node 5.9 (empirical behavior) build on.

Without 5.6 and 5.7 in hand, the cosine schedule of 5.8 would look like an arbitrary trick. With them, it becomes a natural choice: a parameter-space penalty that anneals to zero, preserving the original optimum.

Relations to Other Concepts

- **Tikhonov regularization** — the classical-statistics name for L2.
- **Levenberg-Marquardt** — adaptive trust-region for nonlinear least squares.
- **Gradient clipping** — common in deep learning, conceptually similar to step-size control.
- **Projected gradient descent** — clipping interpreted as projection onto a feasible set.
- **HOPSO damping** — see Module 6; structural parameter-space regularization built into the optimizer.

Worked Example — L2 with annealing schedule

Consider $\lambda_k = \lambda_0 \cdot s(k)$ with $s(k) \to 0$. The update rule

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\big[\nabla C(\boldsymbol{\theta}_k) + \lambda_k \boldsymbol{\theta}_k\big]$$

acts as a damped oscillator early (when λ_k is large) and reduces to vanilla gradient descent late (when $\lambda_k \to 0$). The asymptotic optimum is unchanged because the regularization vanishes; only the **trajectory** is altered. This is the dynamical-modification view in action — see node 5.5 for the formal limit ordering.

Visualization

Pending. Diagram: side-by-side trajectories on a 2D noisy quadratic. Left: vanilla SGD with high variance. Right: SGD + L2 with annealing — same final point, smoother path.

Exercises

1. Show that adding a constant L2 penalty $\lambda\|\boldsymbol{\theta}\|_2^2$ to the cost shifts the minimum away from the unregularized optimum, but a vanishing schedule $\lambda_k \to 0$ does not.
2. Derive the update rule for trust-region gradient descent on a quadratic. What happens when the trust-region radius is much smaller than the gradient magnitude?
3. (VQA) Suppose your parameters live on a torus $[-\pi, \pi]^d$. Design a parameter-space regularizer that respects the periodicity. (Hint: think about what "small" should mean on a circle.)

Research Extensions

- **Manifold-aware regularization** — penalizing motion in the geometry of the Bures or Fubini-Study metric (links back to node 5.2).
- **Adaptive trust regions for noisy oracles** — adjusting Δ_k based on the variance of the cost estimate.
- **Implicit regularization of momentum** — even without explicit penalty, momentum-based optimizers exhibit a regularizing effect on the trajectory.

Cross-links to Other Courses

- **Optimization Theory** — trust-region methods, projected gradient, Levenberg-Marquardt.
- **Math-Intuition** — Lagrangian formulation of constrained optimization.
- **Statistics Module 6** — Bayesian view of the (now-rejected) L2-as-Gaussian-prior mapping (see node 5.1 for the lens, 5.5 for why we reject it in VQA).

Objective-Space Regularization Techniques

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.7

This node catalogs the **objective-space** family of regularization techniques: methods that modify the cost signal $C(\boldsymbol{\theta})$ **before** it reaches the optimizer, without touching the parameter update rule. It is the sibling of node 5.6 and the second operational half of the dynamical-modification view from node 5.5.

The Common Mechanism

The optimizer sees not the raw cost $C(\boldsymbol{\theta}_k)$ but a transformed signal

$$\widetilde{C}_k = \mathcal{F}\big(C(\boldsymbol{\theta}_k), C(\boldsymbol{\theta}_{k-1}), \ldots; \lambda\big).$$

The transformation $\mathcal{F}$ might smooth, weight, or augment the cost signal. The optimizer's update rule is **unchanged** — it still consumes a number called "the cost" — but the number it consumes has been filtered.

This contrast with node 5.6 is the entire point: same optimizer, different feedback.

Technique 1 — Temporal Smoothing

Replace the instantaneous cost with a moving average or exponentially-weighted average:

$$\widetilde{C}_k = (1-\beta)\widetilde{C}_{k-1} + \beta\, C(\boldsymbol{\theta}_k).$$

This introduces inertia into the feedback loop. Transient shot-noise spikes are damped before they can cause large parameter excursions. The price is delayed responsiveness: a genuine improvement in C takes a few iterations to show up in $\widetilde{C}$.

In VQA this is one of the cheapest stabilizers available — no extra circuit evaluations, just a running average — and it directly attacks the dominant noise source on NISQ hardware.

Technique 2 — Variance Penalization

Penalize cost evaluations with high statistical uncertainty:

$$\widetilde{C}(\boldsymbol{\theta}) = C(\boldsymbol{\theta}) + \mu\, \mathrm{Var}\big[\hat{C}(\boldsymbol{\theta})\big].$$

This biases the optimizer toward regions where the cost can be **trusted**, not just where it is small. In VQA the variance of a Pauli expectation estimate depends on $\langle\psi(\boldsymbol{\theta})|H^2|\psi(\boldsymbol{\theta})\rangle - \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle^2$, so this is a real, measurable quantity — not an abstract penalty.

As with all objective-space techniques in VQA, the penalty is typically scheduled to vanish near convergence so the location of the optimum is preserved.

Technique 3 — Shot-Adaptive Weighting

Cost evaluations carry different reliabilities depending on how many measurement shots were used to estimate them. Weight each evaluation by its precision:

$$\widetilde{C}_k = \frac{\sum_j w_j C_j}{\sum_j w_j}, \qquad w_j \propto \frac{1}{\sigma_j^2}.$$

This lets the optimizer use cheap, high-variance estimates exploratorily while still anchoring on rare high-shot evaluations near promising regions.

The resource-allocation question — *how to spend a finite shot budget across iterations* — is where this technique shades into the larger topic of measurement-optimizer co-design (see Module 10).

Technique 4 — Auxiliary Penalty on Cost-Change Magnitude

Penalize rapid swings in successive cost values:

$$\widetilde{C}_k = C(\boldsymbol{\theta}_k) + \nu \left(C(\boldsymbol{\theta}_k) - C(\boldsymbol{\theta}_{k-1})\right)^2.$$

This discourages oscillatory behavior driven by noise. Like temporal smoothing, it adds inertia, but here the inertia is on the **cost signal's volatility** rather than its level.

Structural Role

Objective-space regularization is the natural intervention point when **the optimizer is fine but the signal is noisy**. In VQA this is the typical failure mode: gradient methods, HOPSO, COBYLA — they all behave reasonably given a clean cost signal. The instability comes from shot noise corrupting the feedback.

This node, together with 5.6, gives the practitioner the full menu of dynamical-modification options. Node 5.8 then shows how to **schedule** these techniques over training so they vanish near convergence.

Relations to Other Concepts

- **Polyak averaging** — averaging iterates is a parameter-space analogue of cost averaging.
- **Kalman filtering** — temporal smoothing as a special case of state estimation under noise.
- **CMA-ES** — uses a form of variance-aware weighting in its sample selection.
- **Importance-weighted gradients** — the precision-weighting trick generalized.

Worked Example — Temporal smoothing as a low-pass filter

Treat the cost iterates $\{C(\boldsymbol{\theta}_k)\}$ as a discrete-time signal. The exponential moving average

$$\widetilde{C}_k = (1-\beta)\widetilde{C}_{k-1} + \beta C_k$$

has the transfer function

$$H(z) = \frac{\beta}{1-(1-\beta)z^{-1}},$$

a first-order low-pass filter with cutoff $\omega_c \approx \beta$ (in radians per iteration). Smaller β means heavier smoothing and slower response. The effective cost landscape seen by the optimizer is the original landscape convolved with this filter — high-frequency shot-noise components are suppressed; low-frequency descent directions pass through.

Visualization

Pending. Diagram: cost time series with high-frequency noise overlaid on a slow descent. Two traces: raw C_k (jagged) and smoothed $\widetilde{C}_k$ (clean). Annotate where the optimizer would have fired a destabilizing update on the raw signal.

Exercises

1. Show that exponential smoothing with parameter β is equivalent to a first-order IIR low-pass filter, and derive its 3 dB cutoff frequency.
2. Suppose the variance of your cost estimate scales as $1/N$ where N is the shot count. Design a shot-adaptive weighting scheme that minimizes the variance of $\widetilde{C}_k$ given a total shot budget $N_{\text{tot}} = \sum_j N_j$. (Hint: Lagrange multipliers.)
3. (VQA) Why does temporal smoothing interact badly with population-based optimizers like vanilla PSO? (Hint: think about how attractor positions get updated.)

Research Extensions

- **Adaptive smoothing** — adjusting β based on observed cost variance.
- **Bayesian optimization with noisy oracles** — explicit uncertainty modeling as objective-space regularization.
- **Joint shot-and-step adaptation** — co-designing the optimizer step size and the shot budget per iteration.

Cross-links to Other Courses

- **Signal Processing** — IIR filters, low-pass design, group delay.
- **Statistics Module 6** — Kalman filters and state estimation.
- **Optimization Dynamics (Module 6)** — the input side of the gradient-flow analysis.

The Cosine Schedule and Two-Stage Protocol

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.8

This node introduces the **two-stage protocol** that operationalizes the dynamical-modification view (node 5.5) using a cosine-annealed regularization schedule. It is the most concrete recommendation in Module 5: a specific recipe with specific hyperparameters that has been used in our own experiments. It depends on 5.5 (the framing), 5.6 (parameter-space techniques), and 5.7 (objective-space techniques).

The Idea in One Sentence

> Apply regularization aggressively early to stabilize trajectories, then anneal it to zero so the final iterates converge to the original (unregularized) optimum.

This is the obvious move once you accept the dynamical-modification view of node 5.5: regularization is for the path, not for the destination. If you keep regularization on at convergence you have changed the problem; if you anneal it away you have only changed the route.

The Schedule

Let T be the total iteration budget. Define a cosine schedule on the regularization weight:

$$\lambda_k = \frac{\lambda_0}{2}\left(1 + \cos\frac{\pi k}{T}\right), \qquad k = 0, 1, \ldots, T.$$

At $k = 0$, $\lambda_k = \lambda_0$. At $k = T$, $\lambda_k = 0$. The decay is smooth and monotone with a flat shoulder at the start (so early training experiences nearly the full regularization) and a flat tail at the end (so the optimizer has time to settle into the unregularized optimum).

Cosine schedules are popular in deep learning for the **learning rate**. Here we apply the same shape to the **regularization weight**, for analogous reasons: early phase wants exploration with stabilization, late phase wants fine-tuning without distortion.

The Two-Stage Protocol

The protocol splits the optimizer's run into two phases:

Stage A — Stabilized exploration. For $k \in [0, T_A]$, run the optimizer with the regularization schedule active. Use a relatively small per-iteration budget ($\texttt{maxiter}_A$ inner steps per outer iteration if the optimizer is hierarchical). The goal here is **not** to reach the minimum. The goal is to reach a stable region of parameter space that is robust to shot noise.

Stage B — Unregularized refinement. For $k \in [T_A, T]$, the schedule has effectively decayed; run the optimizer with regularization disabled (or near-zero). Use a longer per-iteration budget ($\texttt{maxiter}_B$). The goal here **is** to reach the minimum. Because Stage A delivered a good initialization, the unregularized phase no longer suffers the instabilities that motivated regularization in the first place.

A Concrete Configuration (Chernyak D30)

The configuration used in our own VQE experiments on a depth-30 problem instance:

Component	Setting
Ansatz	`TwoLocal(rotation_blocks=['rz','ry'], reps=4)`
Parameters	40
Optimizer	HOPSO (see Module 6)
Stage A iterations T_A	15
Stage A inner budget $\texttt{maxiter}_A$	15
Stage B inner budget $\texttt{maxiter}_B$	10
Schedule	cosine on regularization weight
λ_0	tuned per instance, typically small

These numbers are not magic. They were chosen empirically on a specific class of Hamiltonians and they reflect a tradeoff between *time spent stabilizing* and *time spent converging*. They are reported here as a worked example, not as a universal recommendation. Node 5.9 (empirical behavior) discusses how performance depends on these knobs.

Why Two Stages and Not One?

A natural alternative is to keep regularization on continuously, anneal slowly, and run a single-phase optimizer. The two-stage version is operationally cleaner for three reasons:

1. **Hyperparameter independence.** The Stage A budget can be tuned for stabilization, the Stage B budget for refinement. They optimize different quantities and you do not have to find a single setting that does both jobs at once.
2. **Logging clarity.** It is easier to read a training curve when the role of each iteration is explicit. "Why did the loss spike?" is answered differently in Stage A (stabilization phase, expected) and Stage B (refinement phase, suspicious).
3. **Optimizer compatibility.** Some optimizers (HOPSO included) have internal state — particle velocities, momentum buffers — that benefit from a clean reset between phases. The two-stage protocol gives you a natural moment to reset.

Structural Role

This node is the **operational climax** of Module 5. Everything in 5.1–5.7 leads here: - 5.1–5.4 establish the standard view and its limits. - 5.5 reframes regularization as dynamical modification. - 5.6 and 5.7 enumerate the levers. - 5.8 picks one specific lever (annealed parameter-space penalty) and shows how to schedule it.

Node 5.9 (empirical behavior under shot noise) then asks the question this node provokes: *does the protocol actually work?*

Relations to Other Concepts

- **Cosine annealing for learning rate** — same shape, different quantity. Loshchilov-Hutter, SGDR.
- **Curriculum learning** — staging the optimization problem from easier to harder.
- **Trust-region adaptation** — Stage B is essentially a relaxed trust region after Stage A has positioned the iterate.

- **Simulated annealing** — temperature schedule analogy; here λ plays the role of an inverse temperature.

Visualization

Pending. Diagram: a training curve with two color-coded segments. Left half: Stage A, jagged but bounded, with the cosine λ_k curve overlaid. Right half: Stage B, smooth descent to a lower asymptote. Annotate T_A, $\lambda_0 \to 0$, and the final unregularized minimum.

Exercises

1. Plot the cosine schedule $\lambda_k = \frac{\lambda_0}{2}(1 + \cos(\pi k/T))$ and verify that $\lambda_0 = \lambda_0$ and $\lambda_T = 0$. What is the iteration at which $\lambda_k = \lambda_0/2$?
2. Suppose Stage A is too short. Predict qualitatively what the training curve looks like and which failure mode dominates.
3. (VQA) Why is it inappropriate to evaluate the *final* energy of a Stage A run as if it were the answer? What metric should you report instead from Stage A?

Research Extensions

- **Adaptive T_A** — switching from Stage A to Stage B when a stability criterion is met instead of at a fixed iteration.
- **Schedule shapes beyond cosine** — linear, polynomial, step. Empirically, the differences are small once the total integral is matched.
- **Multi-stage protocols** — three or more phases for hierarchical optimization problems.

Cross-links to Other Courses

- **Optimization Dynamics (Module 6)** — interaction with HOPSO's intrinsic damping.
- **Hardware Noise Models (Module 9)** — Stage A's stabilization is most beneficial under high noise.
- **Math-Intuition** — the cosine schedule as a smooth interpolant between two regimes.

Empirical Behavior Under Shot Noise

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Regularization **Node:** 5.9

This is the **closing node** of Module 5. It asks the empirical question that nodes 5.5–5.8 set up: *under realistic shot noise, does the dynamical-modification view of regularization actually deliver more stable, more reliable variational optimization than the unregularized baseline?*

The answer is "yes, in specific regimes," and this node makes the regimes precise.

Why This Is the Right Question

Module 5's argument has been theoretical:

- Standard prior-injection (node 5.1) is the wrong frame for VQA because the optimizer never reaches MAP (node 5.5).
- Regularization should be understood as modifying the **trajectory**, not the **target** (5.5).
- This unlocks specific tools: parameter-space techniques (5.6), objective-space techniques (5.7), and the two-stage protocol with cosine annealing (5.8).

A theoretical argument is necessary but not sufficient. The cash value is whether running real VQE instances with the protocol gives better outcomes than running them without. "Better" here means: lower variance across seeds, more reliable convergence, comparable or better final energies, and improved measurement efficiency.

Experimental Setup

The setup mirrors the empirical chapters of the underlying thesis (see node 5.5 for the framing).

- **Problem class:** variational quantum eigensolver instances on fixed Hamiltonians.
- **Ansatz:** hardware-efficient `TwoLocal` circuits, varying depth and qubit count.
- **Noise model:** finite-shot sampling noise on every cost evaluation.
- **Optimizer:** HOPSO (see Module 6) with and without the two-stage protocol.
- **Replicates:** multiple independent random seeds per condition.

The independent variable is the regularization configuration (off / parameter-space only / objective-space only / two-stage protocol). The dependent variables are stability and efficiency metrics defined below.

Metrics

Convergence speed alone is the wrong yardstick under stochastic noise. The relevant metrics are:

1. **Best-energy variance across seeds.** How much does the final reported energy depend on initialization? Low variance means the protocol is reliable; high variance means the optimizer is finding different local minima or failing in seed-dependent ways.
2. **Worst-case run quality.** The 90th-percentile worst seed, not the median seed, is what determines whether you can trust a single production run.
3. **Iteration-vs-energy trajectory variance.** Even within a single seed, how jittery is the descent? A noisy trajectory is a sign of unstable feedback.

4. **Measurement-budget efficiency.** Energy as a function of total shots consumed, not iteration count. In NISQ, shots are the resource; iterations are an internal accounting unit.

Expected Results (Predictive)

Note: results pending the next vqe-lab sweep. The predictions below come from the dynamical analysis of nodes 5.5–5.8 and from preliminary observations on smaller instances.

Prediction 1. Under high shot noise, the two-stage protocol reduces best-energy variance across seeds compared to the unregularized baseline.

Prediction 2. Under low shot noise, the protocol confers little or no benefit and may slightly slow convergence due to the Stage A overhead.

Prediction 3. Parameter-space techniques (5.6) help most when instability comes from large parameter excursions; objective-space techniques (5.7) help most when instability comes from cost-signal volatility. The two-stage protocol blends both.

Prediction 4. The improvement in **worst-case** seeds is larger than the improvement in **median** seeds. This is the practically important effect: regularization tightens the distribution rather than shifting its center.

Prediction 5. Measurement-budget efficiency (metric 4) improves under the protocol whenever stability improves, because fewer wasted iterations chasing noise-induced bad updates means fewer wasted shots.

What a Negative Result Would Mean

If the protocol fails to improve any of the metrics, the dynamical-modification framing is not falsified — but its operational utility is. The framing would still be **correct** as a description of what regularization does in VQA; it would just turn out that under the specific noise levels and ansatz classes tested, the standard unregularized optimizer is already good enough.

In that case the recommendation would shift: skip the two-stage protocol in low-noise regimes, retain it as a tool for high-noise hardware-on-loop runs. This is a more nuanced position than "the protocol works" or "the protocol fails."

Structural Role

This is the empirical capstone of Module 5. It is also the **bridge** to Module 6 (Optimization Dynamics), where HOPSO is treated in full and the interaction between structured optimizer dynamics and regularization is analyzed. Many of the regimes in which regularization helps are regimes where HOPSO already does part of the job — the question of how the two interact is the natural next step.

Relations to Other Concepts

- **A/B testing under stochasticity** — how to compare two optimizers when their outputs are random variables.
- **Reproducibility in VQA literature** — many published VQE results are single-seed; this section argues for distributional reporting.
- **Measurement-optimizer co-design** — Module 10 picks this up.

Visualization

Pending. Diagram set: 1. *Box-and-whisker plot of final energies across seeds, four conditions side by side.* 2. *Energy-vs-shots curves with confidence bands, showing the protocol's narrower band.* 3. *Per-seed trajectories overlaid, illustrating the variance reduction visually.*

Exercises

1. Design a statistical test for whether two optimizers give significantly different best-energy distributions, given 20 seeds per condition. What is the appropriate test and why?
2. Suppose the protocol improves the median final energy by 1% but reduces the 90th-percentile-worst final energy by 10%. Which result is more important for production deployment, and why?
3. (VQA) Why is *iteration count* a misleading x-axis when comparing optimizers in the shot-noisy regime? What should the x-axis be instead?

Research Extensions

- **Joint shot-and-regularization scheduling** — co-adapting the shot budget and the regularization weight per iteration.
- **Hardware-on-loop validation** — running the protocol on real quantum hardware, where noise is correlated and time-varying rather than i.i.d.
- **Generalization to other ansatz families** — the protocol was tuned on `TwoLocal`; how transferable is it to UCCSD, ADAPT-VQE, or QAOA?

Cross-links to Other Courses

- **Statistics Module 8** — hypothesis testing for stochastic optimizers.
- **Optimization Dynamics (Module 6)** — the next chapter in the same story.
- **Hardware Noise Models (Module 9)** — the source of the noise this node fights against.

Module 6 — Optimization Dynamics

Gradient Descent in VQA

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.1

This is the **opening node** of Module 6 — the module on optimization dynamics. With cost landscapes characterized (Module 3) and barren plateaus understood (Module 4), the question now becomes: **how do we actually move parameters to reduce the cost?**

The simplest answer, and the historical starting point of VQA optimization, is **gradient descent**: at each step, take a small move in the direction of steepest descent. This node walks through what gradient descent looks like in the VQA context, what makes it work, what makes it fail, and how it sets the baseline that all other optimizers are compared against.

The Update Rule

Vanilla gradient descent updates parameters according to

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \, \nabla C(\boldsymbol{\theta}_k),$$

where $\eta > 0$ is the **learning rate** (or step size).

For VQA, the gradient is estimated by parameter-shift (Section 3.8), so each step requires $2P$ cost evaluations for a P-parameter ansatz. Each cost evaluation is itself a noisy quantity (Section 3.7), so what the optimizer actually computes is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \, \widehat{\nabla C}(\boldsymbol{\theta}_k),$$

where $\widehat{\nabla C}$ is a stochastic estimator with mean equal to the true gradient and variance set by the shot budget.

This is **stochastic gradient descent (SGD)** in the strict sense — the gradient is unbiased but noisy. The convergence theory is the textbook stochastic-approximation theory (Robbins-Monro 1951), which Section 6.5 will treat in more detail.

The Three Hyperparameters

A naive gradient descent on a VQA has three knobs:

1. Learning rate η. Too small: convergence is slow. Too large: the optimizer overshoots minima and oscillates or diverges. Optimal η depends on the local Hessian (Section 3.5) — specifically, you need $\eta < 2/\lambda_{\max}$ where $\lambda_{\max}$ is the largest Hessian eigenvalue.

2. Shot budget per evaluation N. Too small: gradient noise overwhelms signal, optimizer does a random walk. Too large: each step costs more than necessary, total budget is wasted on precision that wasn't needed.

3. Stopping criterion. When to halt. Common choices: cost stops decreasing, gradient norm below a threshold, fixed budget exhausted.

These three are interrelated in non-obvious ways. For example, doubling N reduces noise by $\sqrt{2}$, which lets you take larger steps (effectively double η) with the same robustness. So "shots per step" and "step size" trade off against each other in a way that affects the overall budget allocation.

The thesis's "stochastic objective generator" framing (Module 1.8) makes this explicit: the optimizer is making decisions under noise, and N sets the noise level it sees. Tuning N is part of the optimizer's job, not just a fixed hyperparameter.

Why Gradient Descent Is The Default

Gradient descent is the default first-attempt optimizer for VQAs for several reasons:

1. It's local. You only need information at the current point — no global landscape knowledge required. This matches the VQA context, where global landscape information is unavailable.

2. It's cheap per step. $2P$ cost evaluations per step, no Hessian, no extra structure. This is the minimum information you can use and still be doing first-order optimization.

3. It has well-understood theory. For convex landscapes, convergence rate is $O(1/k)$. For L-smooth landscapes, GD with $\eta < 1/L$ is guaranteed to monotonically decrease cost (in expectation).

4. It's a clean baseline. Every fancier optimizer is "GD plus something." You can ask whether the "plus something" is worth the cost. This makes GD the reference point.

5. It works on small problems. For $n < 10$ qubits and shallow ansätze, gradient descent on a VQE problem typically converges in $O(100 - 1000)$ iterations to reasonable accuracy. It's not the fastest possible optimizer, but it's adequate.

Why Gradient Descent Often Fails On VQA

The list of GD's pathologies in VQA is long:

1. Barren plateaus (Module 4). At random initialization in a deep ansatz, the gradient is exponentially small in qubit count. GD makes essentially zero progress per step — the noise dominates the signal. No amount of patience helps; GD does a random walk.

2. Step size sensitivity. The optimal step size depends on the local Hessian, which varies wildly across parameter space. A single global η is rarely optimal everywhere. Too large: oscillations. Too small: glacial progress in the flat regions.

3. Curvature mismatch. GD uses the same step size in all directions. If the landscape has very different curvature in different directions (long, narrow valleys), GD zig-zags inefficiently. This is the "ill-conditioned Hessian" problem from classical optimization.

4. Coordinate-system dependence. GD's behavior depends on how you parameterize the ansatz. If you reparameterize (e.g., $\theta \to 2\theta$), the trajectory changes in a non-trivial way. This is "lack of reparameterization invariance" and it is the central problem the **natural gradient** (Section 6.2) addresses.

5. No escape from local minima. GD converges to whatever critical point is closest to the initialization. For non-convex VQA landscapes (which are most of them), this means GD is at the mercy of the starting point.

6. Stagnation in noise. Once the gradient drops below the shot noise floor, GD effectively stops. Even if there is a meaningful descent direction further away, GD can't get there because it has lost the signal.

These failure modes motivate every other optimizer in Module 6. Quantum natural gradient (6.2) addresses #4. Adam (6.3) addresses #2 and #3. Stochastic optimization theory (6.4, 6.5) addresses #6. Population methods (6.7-6.8) address #5.

A Cautionary Worked Example

Consider the simplest possible VQA: a single qubit, $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$, $H = Z$, $C(\theta) = \cos(\theta)$, with global minimum at $\theta = \pi$.

Setup. Start at $\theta_0 = 0.1$. Learning rate $\eta = 0.5$. $N = 100$ shots per evaluation.

Run. - $k = 0$: $C(0.1) \approx 0.995$. Gradient $\approx -\sin(0.1) = -0.0998$. Update: $\theta_1 = 0.1 + 0.5 \cdot 0.0998 = 0.150$. - $k = 1$: $C(0.15) \approx 0.989$. Gradient ≈ -0.149. Update: $\theta_2 = 0.15 + 0.075 = 0.225$. - $k = 2$: $C(0.225) \approx 0.975$. Gradient ≈ -0.223. Update: $\theta_3 = 0.225 + 0.111 = 0.336$.

The optimizer is making progress, but slowly. The gradient grows as it moves away from the starting point (because $C''(\theta) = -\cos(\theta)$ is positive in this region, i.e., the landscape is convex), so successive steps get larger.

Eventually it crosses $\theta = \pi/2$ where the gradient peaks at magnitude 1 (the inflection point), and then enters the descending side of the cosine. At $\theta \approx \pi$, the gradient is again ~ 0 and the optimizer slows down — this is approaching the minimum. With $\eta = 0.5$, the optimizer converges to within 10^{-3} of π in about 30 iterations.

Now repeat with shot noise. $N = 100$ gives gradient noise of order $\sin(\theta)/\sqrt{100} = 0.1\sin(\theta)$. When the gradient is ~ 0.1 (early or late in the trajectory), the noise is comparable to the signal and the optimizer's progress becomes erratic. At the minimum, the gradient and noise are both ~ 0.001, so the optimizer wanders around the minimum forever rather than converging exactly.

Now scale to 10 qubits with random init in a 5-layer HEA. Initial gradient is $\sim 10^{-2}$. Shot noise at $N = 100$ is ~ 0.1. Signal is below noise. **GD does a random walk and never converges.**

This progression — works in 1D, breaks at 10 qubits — is the canonical reason VQA optimizers need to do something more sophisticated than vanilla gradient descent.

What Gradient Descent Looks Like To The Stochastic Objective Generator

In the thesis's framing (Module 1.8), the optimizer interacts with a stochastic objective generator:

- The generator takes a parameter vector $\boldsymbol{\theta}$ as input.
- It returns a noisy scalar $\widehat{C}(\boldsymbol{\theta})$.
- The generator's properties (variance structure, gradient signal, plateau regions) are determined upstream by the ansatz, the Hamiltonian, the encoding, the noise model, and the shot budget.

Gradient descent's job, in this framing, is to **make decisions about where to query next** based on a sequence of past noisy evaluations. Vanilla GD is a particularly simple decision rule: query at $\boldsymbol{\theta}_k - \eta\widehat{\nabla C}$.

This framing makes clear that GD is **agnostic to the structure of the noise** and to the specific properties of the VQA. It treats the cost function as a black box. Optimizers that exploit VQA-specific structure (parameter-shift, periodicity, sinusoidal cost dependence, QFIM) can do better — and that is what the rest of Module 6 develops.

Structural Role

This is the **baseline** node of Module 6. It defines the simplest first-order optimizer, identifies its failure modes, and sets up the rest of the module as a sequence of refinements that address those failures.

It also forwards directly to Module 5 (regularization), where the same gradient descent dynamics are augmented with regularization terms that encode prior knowledge or shape the optimizer's path.

Relations to Other Concepts

- **Robbins-Monro 1951** — the foundational paper on stochastic approximation, the parent theory.
- **Cauchy 1847** — the original gradient descent paper.
- **Convex optimization** — the regime where GD has clean convergence guarantees.
- **L-smoothness** — the standard assumption needed for GD convergence proofs.
- **Module 5 regularization** — the dynamics-side framework that builds on top of GD.

Worked Example — A Two-Parameter VQE

Consider a 2-qubit VQE with ansatz $R_Y(\theta_1) \otimes R_Y(\theta_2) \cdot \mathrm{CNOT}_{0,1} \cdot |00\rangle$, Hamiltonian $H = Z_0 Z_1 + 0.5 X_0 + 0.5 X_1$.

Cost function (after a tedious but standard expansion):

$$C(\theta_1, \theta_2) = \cos(\theta_1)\cos(\theta_2) + 0.5\sin(\theta_1) + 0.5\sin(\theta_2).$$

Gradients:

$$\frac{\partial C}{\partial \theta_1} = -\sin(\theta_1)\cos(\theta_2) + 0.5\cos(\theta_1).$$
$$\frac{\partial C}{\partial \theta_2} = -\cos(\theta_1)\sin(\theta_2) + 0.5\cos(\theta_2).$$

Setting both gradients to zero: critical points at the solutions of $\tan(\theta_1)\cos(\theta_2) = 0.5$ and $\cos(\theta_1)\tan(\theta_2) = 0.5$.

Numerically, one minimum is at approximately $(\theta_1, \theta_2) \approx (-0.46, -0.46)$ with cost ≈ -1.12.

Run GD from $\boldsymbol{\theta}_0 = (0.5, 0.5)$ with $\eta = 0.1$, no noise: - Step 0: gradient $\approx (0.0, 0.0)$ — wait, let me reconsider. At $(0.5, 0.5)$, $-\sin(0.5)\cos(0.5) + 0.5\cos(0.5) \approx -0.421 + 0.439 = 0.018$. So gradient is small but nonzero. - The optimizer slowly drifts toward $(-0.46, -0.46)$ over ~ 50 steps.

Now with shot noise at $N = 1000$ per cost evaluation: gradient noise is ~ 0.03, comparable to the gradient near the start. The optimizer makes erratic progress for the first ~ 100 steps, then settles into a slow descent once the gradient is large enough.

This 2-parameter example shows the basic GD behavior in miniature: convergence works but is sensitive to noise, particularly when the gradient is small compared to the noise level.

Visualization

Pending. A 2D contour plot of $C(\theta_1, \theta_2)$ with a GD trajectory overlaid. Two trajectories shown: one noiseless (smooth curve to the minimum) and one noisy (erratic walk that eventually finds the minimum). Caption: "GD: simple, slow, noise-limited."

Exercises

1. For a 1D cost $C(\theta) = \theta^2$, show that GD with $\eta < 1$ converges, with $\eta = 1$ reaches the minimum in one step, and with $\eta > 2$ diverges. What is the equivalent statement for a 2D quadratic?
2. For shot-noisy GD on $C(\theta) = \cos(\theta)$ starting at $\theta_0 = 0.1$, estimate the shot budget needed to reach $\theta = \pi$ within 10^{-2} precision. How does this scale with N?

3. (VQA) For a 5-qubit HEA with 3 layers and a global $H = \sum_i Z_i$, estimate the typical gradient magnitude and the shot budget needed for GD to make meaningful progress. At what qubit count does GD become infeasible?

Research Extensions

- **Adaptive shot scheduling for GD** — protocols that allocate more shots when the gradient is small and fewer when it is large.
- **Line-search GD for VQA** — using a backtracking line search to choose η per step.
- **Restart strategies** — when to restart GD with a different initialization to avoid local minima.
- **GD with momentum vs vanilla GD** — quantitative comparison on standard VQA benchmarks.

Cross-links to Other Courses

- **Numerical Optimization** — gradient descent, line search, convergence theory.
- **Stochastic Approximation** — Robbins-Monro, Kiefer-Wolfowitz.
- **Module 3 (Cost Landscapes)** — the geometry GD navigates.
- **Module 4 (Barren Plateaus)** — the failure mode that breaks GD on large problems.
- **Module 5 (Regularization)** — the dynamics-side framework GD lives inside.

Quantum Natural Gradient

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.2

The vanilla gradient descent of node 6.1 has a glaring conceptual problem: **it depends on how you parameterize the ansatz**.

If you reparameterize $\theta \to 2\theta$ for some parameter, the gradient component scales by $1/2$ (chain rule), but the optimal step doesn't. Vanilla GD takes the wrong-sized step under reparameterization. For physically equivalent ansätze with different parameterizations, GD produces different trajectories.

This is unsatisfying. The **natural gradient** is the fix: rescale the gradient by a metric on parameter space, chosen so that the resulting update is invariant under reparameterization. For VQAs, that metric is the **quantum Fisher information matrix (QFIM)**, also known as the Fubini-Study metric, also discussed in node 2.9 as a property of the ansatz manifold.

This node walks through what natural gradient means, why it's the "right" geometry for VQAs, and what it costs to compute.

The Update Rule

Natural gradient descent uses the update

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\, g^{-1}(\boldsymbol{\theta}_k)\, \nabla C(\boldsymbol{\theta}_k),$$

where $g(\boldsymbol{\theta})$ is the QFIM at point $\boldsymbol{\theta}$.

Compare to vanilla GD:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\, I\, \nabla C(\boldsymbol{\theta}_k).$$

The only difference is replacing the identity I with g^{-1}. But this difference matters: it converts the update from a step in *parameter coordinates* to a step in *state space distance*.

What The QFIM Is

The QFIM is the Riemannian metric on the manifold of pure states, pulled back through the parameterization:

$$g_{ij}(\boldsymbol{\theta}) = 4\,\mathrm{Re}\left[\langle \partial_i\psi|\partial_j\psi\rangle - \langle \partial_i\psi|\psi\rangle\langle\psi|\partial_j\psi\rangle\right].$$

Equivalently, it is the Hessian of the **quantum infidelity** $1 - |\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta}+\boldsymbol{\delta})\rangle|^2$ at $\boldsymbol{\delta} = 0$, divided by 2.

Key properties:

- **Positive semidefinite.** $g \geq 0$.
- **Reparameterization-covariant.** If $\boldsymbol{\theta} \to \boldsymbol{\phi}(\boldsymbol{\theta})$, then $g \to J^T g J$ where J is the Jacobian. The combination $g^{-1}\nabla C$ is invariant.
- **Captures state distinguishability.** Two parameter directions with small g_{ii} produce nearly indistinguishable states; large g_{ii} means easily distinguishable.
- **Has a kernel for gauge-redundant ansätze.** If the parameterization has continuous gauge directions (Section 3.4), the QFIM has zero eigenvalues in those directions, and g^{-1} doesn't exist.

The kernel issue is technically inconvenient and is usually handled by replacing g^{-1} with the **Moore-Penrose pseudoinverse** g^{+}.

Why Natural Gradient Is "Natural"

The intuition is that **vanilla GD treats parameter space as flat**, but the parameterization of the ansatz manifold doesn't preserve flatness in any meaningful way.

For example, if θ_1 and θ_2 are two parameters, vanilla GD assumes that "moving by δ in θ_1" and "moving by δ in θ_2" produce equal-sized state-space changes. This is rarely true. θ_1 might be a parameter that strongly affects the state (a single-qubit rotation on a superposition state), while θ_2 is a parameter with weak effect (a rotation on an eigenstate).

Natural gradient corrects for this by **measuring step size in state space, not parameter space**. The QFIM is the conversion factor.

The result: natural gradient takes uniform steps in the *physically meaningful* metric — Fubini-Study distance on the manifold of states. Two trajectories that produce the same sequence of states but use different parameter labels look identical to natural gradient.

This is the same idea as Fisher's natural gradient in statistics (Amari 1998), which uses the classical Fisher information matrix to make gradient descent on probability distributions invariant under reparameterization. The QFIM is the quantum version.

How To Compute The QFIM

For a parameterized circuit, the QFIM has $P \times P$ entries. Computing one entry requires evaluating two quantum-state inner products:

$$g_{ij} = 4\,\mathrm{Re}\left[\langle\partial_i\psi|\partial_j\psi\rangle - \langle\partial_i\psi|\psi\rangle\langle\psi|\partial_j\psi\rangle\right].$$

Each inner product can be evaluated with a **Hadamard test** circuit, which takes 1 ancilla qubit and a controlled version of the ansatz. So computing the full QFIM takes $O(P^2)$ circuit evaluations, each $O(N)$ shots — comparable to computing a Hessian.

For $P = 50$ and $N = 1000$, the QFIM takes $\sim 10^6$ shots per iteration. This is **5-10x more expensive** than computing the gradient itself, which limits how often you can afford to update the QFIM in practice.

Common cost-saving approximations:

1. **Block-diagonal QFIM.** Only compute the diagonal blocks corresponding to layers or qubits, ignoring cross-block correlations. Cost drops from $O(P^2)$ to $O(P \cdot \text{block-size})$.
2. **Diagonal QFIM.** Only compute g_{ii}, no off-diagonal entries. Cost drops to $O(P)$, the same as the gradient. The resulting "natural gradient" is no longer fully invariant but captures the per-coordinate scaling.
3. **Cached QFIM.** Compute the QFIM every K iterations rather than every iteration. Useful when the QFIM doesn't change much between steps.
4. **K-FAC-style approximations.** Borrowed from classical natural gradient literature: factor the QFIM into smaller, cheaper-to-invert pieces.

In practice, block-diagonal or diagonal QFIM is what most VQA implementations use. The fully invariant natural gradient is theoretically appealing but operationally expensive.

What Natural Gradient Buys You

The empirical case for natural gradient:

1. Faster convergence near critical points. At a minimum, the QFIM rescales the gradient so that all directions are equivalent. This eliminates the "ill-conditioning" problem of vanilla GD on stretched-out minima.

2. Better behavior in the barren plateau regime. In a plateau, the QFIM eigenvalues are typically very small (the state barely changes with parameter changes). Inverting the QFIM amplifies these directions. This can, in principle, recover signal from regions where vanilla GD has none — though in practice, the noise on the QFIM estimation is *also* large, so the amplification mostly amplifies noise.

3. Reparameterization invariance. Two ansätze with different parameter labels but the same physics give identical natural-gradient trajectories. This is a clean conceptual property and useful for fair comparison of ansatz designs.

4. Better step-size selection. With the QFIM, the appropriate step size is in *state-space units* (e.g., "move 0.01 in Fubini-Study distance"), which is more interpretable than parameter-space units.

5. Smoother optimization paths. Natural gradient produces trajectories that are smoother in state space — fewer zig-zags, more direct routes to the minimum. This reduces the number of steps needed.

What Natural Gradient Doesn't Buy You

It is important to be honest about what natural gradient *doesn't* fix:

1. Barren plateaus. Natural gradient doesn't eliminate them. If the gradient is below the shot noise floor, multiplying by g^{-1} (whose entries are also exponentially small) doesn't recover signal. The ratio $\nabla C/g$ might be more useful conceptually, but estimating both numerator and denominator in the noise floor regime is hopeless.

2. Local minima. Natural gradient is still a local first-order optimizer. It finds the critical point closest to the initialization. No global escape.

3. Cost. The QFIM is expensive. For $P > 100$ parameters, computing the full QFIM each iteration is prohibitive. The savings from "fewer iterations" don't always outweigh the "more shots per iteration."

In practice, natural gradient is most useful for **medium-sized VQAs** (10-50 parameters) where the shot cost of the QFIM is tolerable and the convergence speedup is meaningful.

The Stoudenmire-Schwab Variant

A specific natural gradient variant worth knowing about: **Stoudenmire-Schwab quantum natural gradient (Stokes et al. 2020)**.

This computes the QFIM efficiently by using the **block structure** of the parameterized ansatz: each ansatz layer depends on a small number of parameters, and the QFIM block for that layer can be computed by Hadamard tests within the layer alone, using the *previous* layer's output as the input state.

For a layered ansatz with L layers and q parameters per layer, the per-layer QFIM block is $q \times q$ and costs $O(q^2)$ circuit evaluations. The total QFIM is approximately block-diagonal with L blocks, costing $O(Lq^2) = O(Pq)$ total — linear in the number of parameters times the layer width.

For typical VQAs with q small (say 5-10 parameters per layer), this brings the QFIM cost down from quadratic to nearly linear in P, making natural gradient practical for ansätze of ~ 100 parameters.

Stokes et al. (2020) showed that this block-diagonal natural gradient gives most of the convergence benefit of the full QFIM at a fraction of the cost. In modern VQA practice, "natural gradient" usually refers to this block-diagonal version unless otherwise specified.

Structural Role

This node is the **geometric** entry point of Module 6. It introduces the QFIM (formally; node 2.9 introduced it informally) and motivates the use of intrinsic geometry rather than ambient parameter coordinates.

It forwards to Section 6.3 (Adam and adaptive methods, which use diagonal-curvature-like rescalings as a poor-man's natural gradient) and Section 6.6 (optimal transport, which generalizes the natural gradient idea to even richer geometries).

Relations to Other Concepts

- **Fisher information (classical)** — Amari's natural gradient.
- **QFIM / Fubini-Study metric** — the quantum analogue.
- **Riemannian optimization** — the general framework for optimization on curved spaces.
- **Newton's method** — second-order optimization that uses the cost Hessian (different from QFIM).
- **Stokes et al. 2020** — the block-diagonal practical QFIM construction.

Worked Example — QFIM For A Single-Qubit Ansatz

Consider $|\psi(\theta_1, \theta_2)\rangle = R_Y(\theta_2)R_X(\theta_1)|0\rangle$.

Compute the partial derivatives:

$$|\partial_1\psi\rangle = -\frac{i}{2}R_Y(\theta_2)XR_X(\theta_1)|0\rangle.$$
$$|\partial_2\psi\rangle = -\frac{i}{2}YR_Y(\theta_2)R_X(\theta_1)|0\rangle.$$

Compute the inner products:

$$\langle\partial_1\psi|\partial_1\psi\rangle = \frac{1}{4}\langle 0|R_X^\dagger X^2 R_X|0\rangle = \frac{1}{4}.$$
$$\langle\partial_2\psi|\partial_2\psi\rangle = \frac{1}{4}\langle\psi(0)|Y^2|\psi(0)\rangle = \frac{1}{4}.$$
$$\langle\partial_1\psi|\partial_2\psi\rangle = -\frac{1}{4}\langle 0|R_X^\dagger X R_Y^\dagger Y R_Y R_X|0\rangle = ?$$

Working out the third quantity gives a θ_1, θ_2-dependent value.

The QFIM is

$$g(\theta_1, \theta_2) = 4\begin{pmatrix} \frac{1}{4} - |\langle\partial_1\psi|\psi\rangle|^2 & \cdots \\ \cdots & \frac{1}{4} - |\langle\partial_2\psi|\psi\rangle|^2 \end{pmatrix}.$$

For $\theta_1 = 0$, the state is $R_Y(\theta_2)|0\rangle = \cos(\theta_2/2)|0\rangle + \sin(\theta_2/2)|1\rangle$, and the QFIM evaluates to

$$g(0, \theta_2) = \begin{pmatrix} \cos^2(\theta_2/2) & 0 \\ 0 & 1 \end{pmatrix}.$$

Note: at $\theta_2 = \pi$, the (1,1) entry vanishes — the QFIM has a kernel here, and natural gradient is undefined in the θ_1 direction. The vanishing happens because at this point, the state $R_Y(\pi)|0\rangle = |1\rangle$ is an eigenstate of X (up to phase), so an infinitesimal rotation around the X-axis doesn't change the state. There is no direction in state space corresponding to motion in θ_1, so natural gradient pseudo-inverse must handle this gracefully.

This is the kind of degeneracy you see in QFIMs in practice; pseudoinverse handles it cleanly.

Visualization

Pending. A 2D parameter-space plot showing two trajectories on the same cost landscape: one from vanilla GD (zig-zag) and one from natural gradient (smoother). Annotations highlight the QFIM rescaling at each step. Caption: "Vanilla GD walks in coordinates. Natural gradient walks in state space."

Exercises

1. For $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$, compute the QFIM (which is just a scalar). Show that it equals 1 for all θ, and explain why.
2. For a 2-parameter ansatz with QFIM $g = \begin{pmatrix} 4 & 1 \\ 1 & 1 \end{pmatrix}$, compare the vanilla and natural gradient updates for a fixed gradient $\nabla C = (1, 1)^T$. Which direction does each take?
3. (VQA) Estimate the cost of a single natural gradient step (in shot count) for a 30-parameter ansatz, using the block-diagonal QFIM with 5 parameters per block. Compare to a vanilla gradient step.

Research Extensions

- **K-FAC and other approximate natural gradients** — borrowed from classical deep learning.
- **Online QFIM estimation** — updating the QFIM incrementally rather than recomputing from scratch.
- **Gauge-invariant natural gradient** — formulations that handle continuous gauge redundancies cleanly.
- **Hardware-aware QFIM** — accounting for noise in the QFIM computation itself.

Cross-links to Other Courses

- **Information Geometry** — Amari's natural gradient and Fisher information.
- **Differential Geometry** — Riemannian metrics, geodesics, parallel transport.
- **Module 2 (Ansatz as Manifold, Section 2.9)** — informal introduction to the QFIM.
- **Module 3 (Local Geometry, Section 3.5)** — gradient and Hessian alongside the QFIM.

Adam and Adaptive Methods

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.3

Vanilla gradient descent (6.1) uses one global learning rate. Natural gradient (6.2) rescales the gradient by the QFIM, which is geometrically principled but expensive. **Adaptive methods** sit in between: they rescale the gradient using *cheap, online estimates* of curvature derived from the gradient history itself, without ever computing the QFIM.

The most famous adaptive method is **Adam** (Kingma and Ba, 2014), which has become the default optimizer in classical deep learning. Variants include RMSProp, AdaGrad, AdaDelta, AMSGrad, and RAdam — all with similar structure but different details.

This node describes the Adam family, explains why they often outperform vanilla GD on VQAs, and identifies their failure modes.

The Adam Update Rule

Adam maintains two moving averages alongside the parameters:

- $\mathbf{m}_k$: exponential moving average of gradients (first moment).
- $\mathbf{v}_k$: exponential moving average of squared gradients (second moment).

At each step:

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\nabla C(\boldsymbol{\theta}_{k-1}),$$
$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)[\nabla C(\boldsymbol{\theta}_{k-1})]^2,$$
$$\widehat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1-\beta_1^k}, \quad \widehat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_2^k} \quad \text{(bias correction)},$$
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \frac{\widehat{\mathbf{m}}_k}{\sqrt{\widehat{\mathbf{v}}_k} + \epsilon}.$$

The default hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\eta = 10^{-3}$.

The squared-gradient division is the key. **It rescales each component of the update by an estimate of the local gradient variance**, which is roughly an estimate of the local curvature in that direction. Components with consistently large gradients get smaller effective steps; components with consistently small gradients get larger effective steps.

Why The Squared-Gradient Rescaling Works

The intuition is: $\mathbf{v}_k$ is an exponential moving average of $|\nabla C|^2$. For a smooth landscape near a minimum, $|\nabla C|^2$ is approximately related to the local curvature: high-curvature directions have rapidly-changing gradients, so $\mathbf{v}$ is large there; low-curvature directions have slowly-changing gradients, so $\mathbf{v}$ is small.

Dividing by $\sqrt{\mathbf{v}}$ thus produces an update where steps are normalized by $\sqrt{\text{curvature}}$, which is similar to what natural gradient does — except the "curvature" estimate is *empirical*, not derived from the QFIM.

This is much cheaper than computing the QFIM (no extra circuit evaluations), and it's adaptive (the estimate updates as the optimizer moves). The cost is that the curvature estimate is *noisy* and *biased* — $\mathbf{v}$ reflects past gradients, which may not reflect current curvature, and it's contaminated by shot noise.

In classical deep learning, this rescaling is usually a net win. In VQAs, the picture is more nuanced (see "Failure Modes" below).

Why Adam Works Well For VQA

1. Per-coordinate learning rates. Different parameters in a VQA ansatz often have wildly different sensitivities. A single global learning rate either underutilizes the small-sensitivity parameters or overshoots on the large-sensitivity ones. Adam's per-coordinate rescaling handles this automatically.

2. Robustness to gradient noise. The first-moment averaging $\mathbf{m}$ acts like momentum, smoothing out noisy gradient estimates over many iterations. This is helpful when shot noise is comparable to the gradient signal.

3. Adaptive step size near minima. As the optimizer approaches a minimum, the gradient shrinks but $\mathbf{v}$ remains large (it's a moving average of past, larger gradients), so the effective step size shrinks gracefully — no manual learning-rate decay needed.

4. No QFIM. Adam approximates curvature information without ever computing the QFIM, saving the $O(P^2)$ cost.

5. Robust default hyperparameters. The defaults ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 10^{-3}$) work reasonably well across a wide range of problems. This makes Adam a "fire and forget" choice for an unfamiliar VQA instance.

For these reasons, Adam is the **most common practical optimizer** for VQAs in 2026 software (Pennylane, Qiskit, Yao). It is the default for "I just want to optimize this thing without thinking about it."

Failure Modes Of Adam On VQAs

Adam's strengths come with corresponding weaknesses:

1. Bias from noisy gradients. $\mathbf{v}$ accumulates squared *noisy* gradients, not squared true gradients. This biases the estimate of curvature upward (because variance contributes to expectation of squared values), which biases the step size *downward*. In the high-noise regime, Adam can be more conservative than necessary, taking smaller steps than vanilla GD with the same learning rate.

2. No barren plateau immunity. Like vanilla GD and natural gradient, Adam doesn't escape barren plateaus. If the gradient is below the noise floor, no rescaling can recover signal.

3. Hyperparameter sensitivity in the noisy regime. The default $\beta_2 = 0.999$ (essentially infinite memory) was chosen for classical deep learning where the gradient is computed once per minibatch on a deterministic loss. For VQAs with shot noise, the optimal β_2 is typically smaller (more weight on recent gradients), and the optimal η depends strongly on the shot budget.

4. Coordinate-system dependence. Adam is *less* coordinate-dependent than vanilla GD (the per-coordinate rescaling helps), but it is *not* fully reparameterization-invariant the way natural gradient is. Reparameterizing the ansatz changes the Adam trajectory.

5. Convergence to non-stationary points (Reddi et al. 2018). Adam was shown to fail to converge in some convex examples — the squared-gradient bias can lead the optimizer to oscillate around a non-minimum point. **AMSGrad** is a variant that fixes this by using the maximum of past $\mathbf{v}$ values rather than an exponential average.

These failure modes are usually mild in practice but worth knowing about.

Variants

RMSProp (Hinton, 2012): Adam without the first-moment averaging. Just $\theta_{k+1} = \theta_k - \eta \nabla C / \sqrt{\mathbf{v}_k}$. Older, simpler, sometimes preferable when you don't want momentum dynamics.

AdaGrad (Duchi et al., 2011): Uses the cumulative sum of squared gradients instead of an exponential average. Step sizes monotonically decrease over time. Good for problems with sparse updates; converges very slowly on dense problems.

AdaDelta (Zeiler, 2012): Like RMSProp but with adaptive learning rate. Largely superseded by Adam.

AMSGrad (Reddi et al., 2018): Fixes Adam's non-convergence by using max over past $\mathbf{v}$ values.

RAdam (Liu et al., 2020): Rectified Adam, addresses the warm-up period instability of Adam.

LAMB / LARS: Layer-wise adaptive methods designed for large-batch training in deep learning. Less relevant for VQAs.

For VQA work, **Adam** and **AMSGrad** are the most commonly used. Other variants are situational.

Tuning Adam For VQA

Learning rate η: typically 10^{-2} to 10^{-1} for VQAs (larger than the classical default of 10^{-3}), because VQA gradients are usually smaller than classical NN gradients due to the cosine-bounded nature of expectation values.

β_2: consider lowering from 0.999 to 0.99 or 0.9 for noisy VQA optimization. The lower β_2 gives more weight to recent gradients, which is helpful when the noise level is high.

Shot budget: Adam's smoothing means you can use fewer shots per evaluation than vanilla GD with the same stability — the moving average does some of the noise reduction for you. N can often be reduced by a factor of 4-10 compared to a vanilla GD baseline with the same learning rate.

Restarts: if Adam stalls, restart with the current parameters but reset $\mathbf{m}$ and $\mathbf{v}$ to zero. This sometimes breaks out of bad regions where the moving averages have accumulated stale information.

These tuning adjustments are not from the original Adam paper but are folk wisdom from VQA practice.

Adam vs Natural Gradient

A natural question: does Adam approximate natural gradient?

Sort of, but not really.

Natural gradient rescales by g^{-1}, the inverse QFIM. Adam rescales by $1/\sqrt{\mathbf{v}}$, the inverse-sqrt of moving-average squared gradients. These are *related* — the squared-gradient moving average is a (noisy, biased) estimate of the diagonal of the cost Hessian, and the cost Hessian is related to the QFIM through the cost function. But they are not the same.

Specifically: - Natural gradient uses **state-space curvature** (independent of cost function). - Adam uses **cost-space curvature estimate** (depends on cost function and gradient history).

In some regimes they agree; in others they differ significantly. Empirically, **Adam is usually faster per step (no QFIM cost) but Natural Gradient is usually fewer steps to convergence** (more accurate descent direction). The total budget winner depends on the problem.

A reasonable rule of thumb: use Adam by default; switch to natural gradient if you're spending too many iterations to converge and the per-iteration QFIM cost is affordable.

Structural Role

This node is the **practical default** of Module 6. Adam is what most VQA implementations use, and understanding its behavior — strengths, failure modes, hyperparameters — is essential for any practitioner.

It also bridges between vanilla GD (6.1) and natural gradient (6.2), occupying a middle ground in the cost-vs-quality tradeoff.

Relations to Other Concepts

- **Kingma and Ba 2014** — the original Adam paper.
- **Reddi et al. 2018** — AMSGrad and the convergence analysis.
- **Momentum methods (Polyak, Nesterov)** — Adam's first-moment averaging is essentially momentum.
- **AdaGrad / RMSProp** — predecessors that motivated Adam.
- **Natural gradient** — the geometric ideal that Adam approximates cheaply.

Worked Example — Adam Vs GD On A Two-Parameter Cost

Consider $C(\theta_1, \theta_2) = \theta_1^2 + 100\theta_2^2$ — a stretched quadratic with very different curvature in the two directions.

Vanilla GD: the optimal step size is $\eta = 1/100 = 0.01$ (limited by the worst direction). At $\boldsymbol{\theta}_0 = (1, 1)$, the gradient is $(2, 200)$. The update is $(1, 1) - 0.01 \cdot (2, 200) = (0.98, -1)$. Notice it overshot $\theta_2 = 0$ and went to -1. The next step will overshoot back. Vanilla GD oscillates in θ_2 while crawling in θ_1.

Adam (with default $\eta = 0.001$): at iteration 1, $\mathbf{m} = (0.2, 20)$ and $\mathbf{v} = (0.004, 40)$. The update is $-0.001 \cdot (0.2/\sqrt{0.004}, 20/\sqrt{40}) = -0.001 \cdot (3.16, 3.16) = (-0.003, -0.003)$. **Both components have the same step size**, despite the very different gradient magnitudes. This is the per-coordinate rescaling at work.

After many iterations, Adam approaches the minimum at roughly equal rates in both directions, while vanilla GD oscillates wildly. Adam reaches a precision of 10^{-3} in ~ 1000 iterations; vanilla GD with the same number of iterations is still oscillating.

For the analogous problem in VQA — a parameter that strongly affects cost (θ_1-like) and one that weakly affects cost (θ_2-like) — Adam handles both gracefully where vanilla GD struggles.

Visualization

Pending. A 2D contour plot of a stretched-quadratic loss with two trajectories: vanilla GD (zig-zag, slow descent in long direction) and Adam (smooth, balanced descent in both directions). Caption: "Adam handles ill-conditioning that GD cannot."

Exercises

1. For the stretched quadratic $C(\theta_1, \theta_2) = a\theta_1^2 + b\theta_2^2$, derive the optimal step size for vanilla GD and compare to Adam's effective step size at each component.
2. The bias correction in Adam (the $1 - \beta^k$ divisions) only matters in early iterations. Show that without bias correction, Adam under-estimates $\mathbf{m}$ and $\mathbf{v}$ for the first $\sim 1/(1 - \beta)$ iterations.
3. (VQA) For a 20-parameter HEA with shot budget $N = 1000$, estimate the relative iteration count for vanilla GD vs Adam to converge to a fixed cost precision. Which uses fewer total shots?

Research Extensions

- **Adam variants for shot-noisy gradients** — versions of Adam that explicitly model shot noise in the moment estimates.
- **Adam + natural gradient hybrids** — using Adam as a fast preconditioner and natural gradient as a refinement.
- **Hyperparameter auto-tuning for Adam on VQA** — meta-learning the optimal β_2 for a given problem class.
- **Catalyst-style acceleration of Adam** — combining Adam with classical acceleration techniques.

Cross-links to Other Courses

- **Deep Learning** — Adam is the workhorse optimizer of classical neural network training.
- **Stochastic Optimization** — convergence theory for noisy gradient methods.
- **Module 6 Section 6.5 (Convergence Theory)** — formal convergence analysis of Adam.
- **Module 5 (Regularization)** — Adam is often combined with regularization terms in the objective.

Stochastic Optimization

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.4

Every VQA optimizer is, by necessity, a **stochastic** optimizer. The cost function it sees is corrupted by shot noise (Section 3.7), the gradient it computes is corrupted by parameter-shift estimation noise, and on real hardware, the gates are corrupted by physical noise. The optimizer's job is not to optimize a deterministic function — it's to make good decisions under noise.

This node introduces the **stochastic optimization** framework that the rest of Module 6 lives inside. It covers the foundational theory (Robbins-Monro), the special class of zero-order methods (Kiefer-Wolfowitz, SPSA), and the practical implications for VQA optimizer design.

The thesis's "stochastic objective generator" abstraction (Module 1.8) is most explicitly cashed out here.

The Stochastic Optimization Setting

A stochastic optimization problem is:

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\xi}[f(\boldsymbol{\theta}, \xi)],$$

where ξ is a random variable representing noise, and the optimizer can only access noisy samples $f(\boldsymbol{\theta}, \xi)$ — not the true expectation.

For VQAs, $\boldsymbol{\theta}$ is the parameter vector, ξ is the randomness from quantum measurement (shot noise) plus hardware noise, and f is the cost-function-with-noise. The expected value $\mathbb{E}_{\xi}[f]$ is the noiseless cost $C(\boldsymbol{\theta})$ (assuming unbiased estimation).

The question is: **how do you find** $\arg\min C$ **when you can only see noisy estimates of** C **and its derivatives?**

This is the territory of stochastic optimization theory, which has been developed since the 1950s and is now a mature field.

Robbins-Monro: The Foundational Result

Robbins and Monro (1951) proved that a simple recursion can find roots of a function from noisy estimates of the function's value:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - a_k \widehat{\nabla C}(\boldsymbol{\theta}_k),$$

where a_k is a sequence of step sizes (the "learning rate schedule") satisfying:

1. $\sum_k a_k = \infty$ — the steps don't decrease too fast.
2. $\sum_k a_k^2 < \infty$ — the steps eventually decrease.

A canonical choice is $a_k = a/k$.

Theorem: Under these conditions, plus mild regularity on C (smoothness, bounded variance of the gradient estimator, unique minimum), the iterates $\boldsymbol{\theta}_k$ converge almost surely to the minimum of C.

Why this matters: Robbins-Monro gives the first formal guarantee that stochastic gradient descent works *despite the noise*, provided the learning rate is decreased appropriately. The decreasing learning rate is the key — it averages out noise asymptotically.

For VQA, the Robbins-Monro condition translates to: - Use a decreasing learning rate, e.g., $\eta_k = \eta_0/k$ or $\eta_0/\sqrt{k}$. - Allocate enough shots that the gradient estimate is unbiased and has finite variance. - Be patient — convergence is asymptotic, and the rate is typically $O(1/\sqrt{k})$ for noisy problems.

Convergence Rates In Stochastic Optimization

For a smooth, strongly convex C with bounded gradient variance σ^2, the convergence rate of stochastic GD is

$$\mathbb{E}[C(\boldsymbol{\theta}_k) - C^*] \leq \frac{\sigma^2}{\mu k} + O\left(\frac{1}{k^2}\right),$$

where μ is the strong-convexity parameter.

For VQAs (which are typically *not* convex), the right notion of convergence is to a critical point, not the global minimum:

$$\mathbb{E}[\|\nabla C(\boldsymbol{\theta}_k)\|^2] \leq \frac{C}{\sqrt{k}}.$$

The $1/\sqrt{k}$ rate is the canonical "stochastic optimization rate" — slower than the $1/k$ of deterministic GD on smooth-convex problems, because the noise contributes a floor on accuracy.

The implication: if you want to halve the gradient norm in stochastic optimization, you need *4x* more iterations, not just 2x. This is why VQA optimizers often need many hundreds or thousands of iterations to converge — the noise scaling is unfavorable.

Mini-Batch SGD And Its Quantum Analogue

In classical deep learning, **mini-batch SGD** estimates the gradient from a subset (mini-batch) of training data, which gives a noisy but unbiased gradient estimate. The mini-batch size trades off computation per step against gradient noise.

The VQA analogue is **shot allocation**: the number of shots per cost evaluation determines the gradient noise level. More shots = lower noise = larger possible step size = fewer iterations. Fewer shots = higher noise = smaller possible step size = more iterations.

The **total shot budget** is the right thing to optimize. Given a fixed total of S shots, you can spend them as: - Many cheap iterations: S/(cost per iter) iterations, each with high noise per iter. - Few expensive iterations: S/(cost per iter) iterations, each with low noise.

The optimal allocation depends on the noise structure. As a rule of thumb, **noisy gradients with many cheap iterations** wins for problems where the optimizer can recover from noise (e.g., convex problems or near a minimum), while **clean gradients with few expensive iterations** wins for problems where noise causes instability (e.g., near a saddle point or in a barren plateau).

This is one of the genuine VQA-specific design problems: choosing the shot budget per evaluation, which has no clean classical analogue.

Zero-Order Stochastic Optimization

What if you don't have gradients at all — only noisy cost evaluations?

This is the **zero-order** setting, and it has its own theory.

Kiefer-Wolfowitz (1952): the original zero-order analogue of Robbins-Monro. Estimates each gradient component by finite differences:

$$\widehat{\partial_i C} = \frac{C(\boldsymbol{\theta} + h\mathbf{e}_i) - C(\boldsymbol{\theta} - h\mathbf{e}_i)}{2h}.$$

Each component requires 2 cost evaluations, so the full gradient costs $2P$ — same as parameter-shift, but with finite-difference error.

SPSA (Spall, 1992): Simultaneous Perturbation Stochastic Approximation. Estimates the *full gradient* from just **2 cost evaluations**, regardless of P. The trick: perturb all parameters simultaneously by a random sign vector $\boldsymbol{\Delta}$, and use the difference $C(\boldsymbol{\theta} + h\boldsymbol{\Delta}) - C(\boldsymbol{\theta} - h\boldsymbol{\Delta})$ to estimate the gradient projected onto the random direction.

The SPSA gradient estimate is

$$\widehat{\partial_i C} = \frac{C(\boldsymbol{\theta} + h\boldsymbol{\Delta}) - C(\boldsymbol{\theta} - h\boldsymbol{\Delta})}{2h\Delta_i}.$$

This is unbiased (in expectation over $\boldsymbol{\Delta}$) and gives a noisy but useful descent direction.

Convergence: SPSA converges at the same asymptotic rate as full-gradient stochastic descent, but with $P/2$ **times fewer cost evaluations per iteration**. For large P, this is a huge saving.

The cost is that the per-iteration gradient is much noisier, so SPSA needs more iterations to reach the same accuracy. The **total cost** can still be lower because the per-iteration cost drops from $O(P)$ to $O(1)$.

For VQAs with $P \sim 100$, SPSA can be 10-50x faster than vanilla parameter-shift gradient descent, especially in the early iterations where high gradient precision is unnecessary.

SPSA In VQA Practice

SPSA was popularized in the VQA community by **Kandala et al. (2017)** ("Hardware-efficient variational quantum eigensolver for small molecules"), which used it to optimize VQE on real superconducting hardware. Since then it has been a default choice for hardware experiments where shot budgets are tight.

Practical SPSA workflow:

1. Choose perturbation size h (typically $h = c/k^{0.101}$, the canonical Spall schedule).
2. Choose learning rate $a_k = a/(k + A)^{0.602}$, with A a small constant.
3. At each iteration, draw $\boldsymbol{\Delta}$ as a vector of ± 1 (Bernoulli rademacher).
4. Evaluate $C(\boldsymbol{\theta} + h\boldsymbol{\Delta})$ and $C(\boldsymbol{\theta} - h\boldsymbol{\Delta})$.
5. Compute the SPSA gradient estimate.
6. Update parameters with $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - a_k \widehat{\nabla C}$.
7. Repeat.

The schedules $h \sim 1/k^{0.101}$ and $a \sim 1/k^{0.602}$ come from Spall's analysis as the asymptotically optimal choices. In practice they are tuned per problem.

When To Use SPSA Vs Parameter-Shift

Use SPSA when:

- You have many parameters ($P > 50$) and the per-iteration cost of parameter-shift is prohibitive.
- The problem is well-conditioned enough that noisy gradients are tolerable.
- You're early in the optimization and want to make fast progress before fine-tuning.
- You're in a regime where parameter-shift's precision is wasted (e.g., early iterations where you just need a descent direction, not an accurate one).

Use parameter-shift when:

- You have few parameters ($P < 30$) and per-iteration cost is manageable.
- You're late in the optimization and need accurate gradients for fine-tuning.
- The problem is ill-conditioned and noisy gradients lead to instability.
- You need exact gradients for theoretical reasons (e.g., natural gradient with QFIM).

A common workflow: use SPSA for the first 100-1000 iterations to get into a good basin, then switch to parameter-shift for fine-tuning.

Other Zero-Order Methods

Coordinate descent: update one parameter at a time using its 1D gradient. Good for problems with structure.

Random search: sample random parameter perturbations, accept those that decrease the cost. Slow but very robust to noise.

Bayesian optimization: model the cost function as a Gaussian process, query points to maximize information gain. Expensive per iteration but data-efficient.

Trust-region methods: maintain a region within which the local approximation is trusted, only step within the region. Good for noisy problems.

These all have niches in VQA optimization, but parameter-shift gradient descent (with some adaptive variant) and SPSA are the dominant choices.

Structural Role

This node is the **theoretical foundation** for everything stochastic in Module 6. It justifies why noisy optimization is possible, gives the convergence rates that subsequent sections use, and introduces SPSA as the principal alternative to parameter-shift gradients.

It also connects directly to the thesis's "stochastic objective generator" abstraction (Module 1.8), which is the conceptual frame for treating the VQA optimizer as a stochastic-decision-maker rather than as a smooth-function-minimizer.

Relations to Other Concepts

- **Robbins-Monro 1951** — the foundational stochastic approximation theorem.
- **Kiefer-Wolfowitz 1952** — the zero-order analogue.
- **Spall 1992** — SPSA, the efficient zero-order method.
- **Stochastic gradient descent (SGD)** — the classical deep learning workhorse.
- **Mini-batch optimization** — the classical analogue of shot allocation.

Worked Example — SPSA On A 4-Parameter VQE

Consider a 4-parameter ansatz with cost $C(\boldsymbol{\theta}) = \sum_{i=1}^{4}(\theta_i - i)^2$, so the minimum is at $\boldsymbol{\theta}^* = (1, 2, 3, 4)$ with $C^* = 0$.

Parameter-shift gradient: at any point, the gradient is exactly $2(\boldsymbol{\theta} - (1, 2, 3, 4))$, and computing it requires 8 cost evaluations.

SPSA gradient at $\boldsymbol{\theta}_0 = (0, 0, 0, 0)$:

Draw $\boldsymbol{\Delta} = (+1, +1, -1, +1)$. Evaluate

$$C(\boldsymbol{\theta}_0 + h\boldsymbol{\Delta}) - C(\boldsymbol{\theta}_0 - h\boldsymbol{\Delta}) = ?$$

With $h = 0.1$: - $C(0.1, 0.1, -0.1, 0.1) = 0.81 + 3.61 + 9.61 + 15.21 = 29.24$. - $C(-0.1, -0.1, 0.1, -0.1) = 1.21 + 4.41 + 8.41 + 16.81 = 30.84$. - Difference: $29.24 - 30.84 = -1.60$.

SPSA gradient estimate:

$$\widehat{\partial_i C} = \frac{-1.60}{2 \cdot 0.1 \cdot \Delta_i} = -8/\Delta_i.$$

- For $i = 1$: $-8/(+1) = -8$. True gradient: $2(0 - 1) = -2$. SPSA estimate is biased high in magnitude but correct in sign. - For $i = 2$: $-8/(+1) = -8$. True: -4. - For $i = 3$: $-8/(-1) = +8$. True: -6. **Wrong sign!** - For $i = 4$: $-8/(+1) = -8$. True: -8. Correct by luck.

The single-iteration SPSA estimate is noisy and can be wrong on individual components. But **averaged over many iterations** (each with a fresh $\boldsymbol{\Delta}$), the estimate converges to the true gradient. In the long run, SPSA produces a valid descent direction with much less per-iteration cost than parameter-shift.

For this 4-parameter problem, SPSA converges in roughly the same number of iterations as parameter-shift, but each iteration costs 2 cost evaluations instead of 8 — a 4x saving. For a 100-parameter problem, the saving is 100x.

Visualization

Pending. Two trajectories on a 2D cost landscape: parameter-shift gradient descent (smooth, slow) and SPSA (jagged, fast). The SPSA trajectory is noisier per step but reaches the minimum in fewer total cost evaluations. Caption: "SPSA: noisy but cheap."

Exercises

1. Verify Robbins-Monro conditions for the schedule $a_k = 1/k$. Verify that they fail for $a_k = 1/k^2$ (sums to a finite value).
2. For the same 4-parameter cost function above, run SPSA for 100 iterations (analytically or in code) and observe how fast the parameters converge to the minimum.
3. (VQA) For a 50-parameter HEA with shot budget $N = 1000$ per evaluation, compare the total shot cost of (a) 1000 parameter-shift iterations and (b) 10000 SPSA iterations. Which uses fewer total shots? Which is likely to give a better final cost?

Research Extensions

- **Variance-reduced SPSA** — combining SPSA with control variates to reduce per-iteration noise.
- **Adaptive perturbation sizes** — choosing h_k based on the local landscape.
- **SPSA + Adam hybrids** — using SPSA gradients in an Adam-like update rule.
- **Quantum-aware SPSA** — exploiting the periodicity and sinusoidal structure of VQA cost functions to improve SPSA bias.

Cross-links to Other Courses

- **Stochastic Approximation Theory** — the broader field.
- **Reinforcement Learning** — uses similar zero-order techniques (REINFORCE, evolution strategies).
- **Module 3 (Shot Noise)** — the noise structure SPSA navigates.
- **Module 6 Section 6.1 (Gradient Descent in VQA)** — the parameter-shift baseline SPSA competes with.

Convergence Theory

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.5

This node is the **theoretical** node of Module 6. It collects the convergence guarantees for VQA optimizers — what we can prove about how fast they converge, under what conditions, and to what.

The honest answer is: **less than we'd like.** Most classical convergence theory assumes convexity or strong-convexity, which VQA cost landscapes do not satisfy. Most stochastic convergence theory assumes bounded-variance noise, which is violated by hardware noise (which can be biased) and is borderline for shot noise. The result is that VQA optimizer behavior is mostly understood empirically, with theory providing a (loose) skeleton.

That said, some theoretical results do apply, and they are worth knowing because they tell you what's *possible* and what's *not* in principle.

The Convergence Hierarchy

VQA convergence theory roughly stratifies into the following hierarchy, from strongest to weakest:

1. **Convex strongly-convex theory:** $O(1/k)$ or $O(\rho^k)$ (linear). Requires convex C. **Does not apply to VQAs in general.**
2. **Smooth non-convex theory:** $O(1/\sqrt{k})$ for $\mathbb{E}[\|\nabla C\|^2]$ under SGD. Requires L-smooth C and bounded gradient variance. **Applies to VQAs in shallow regimes.**
3. **PL-condition theory:** $O(\rho^k)$ under the Polyak-Łojasiewicz condition (a weakened convexity replacement). **Sometimes applies to VQAs near minima.**
4. **No-guarantee theory:** convergence is empirical, no formal rates. **The default assumption for VQAs in practice.**

The challenge of VQA convergence theory is figuring out which level of the hierarchy applies to which VQA instance, and what the relevant constants are.

L-Smoothness And Why It Holds For VQAs

A cost function is L**-smooth** if its gradient is L-Lipschitz:

$$\|\nabla C(\boldsymbol{\theta}_1) - \nabla C(\boldsymbol{\theta}_2)\| \leq L\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|.$$

For VQAs, this is **always satisfied** with a calculable L, because the cost is built from cosines and sines of parameters, which have bounded second derivatives.

Specifically: for an ansatz built from Pauli rotations, each parameter contributes at most $\|H\|$ to the Lipschitz constant of the gradient (this is a consequence of the parameter-shift identity and the boundedness of H). For a P-parameter ansatz, $L \leq \|H\| \cdot P$ at most, often much less.

L-smoothness is what enables the standard SGD convergence rate $O(1/\sqrt{k})$ for non-convex objectives. This is the **only** clean convergence rate that applies to VQAs in general — it gives you a guarantee that, asymptotically, you're heading toward a critical point.

The catch: $O(1/\sqrt{k})$ for the *gradient norm*, not the cost. To convert "small gradient" into "small cost minus optimum" requires additional structure (convexity or PL).

The Polyak-Łojasiewicz (PL) Condition

The PL condition is a weakened form of convexity that often holds locally:

$$\frac{1}{2}\|\nabla C(\boldsymbol{\theta})\|^2 \geq \mu(C(\boldsymbol{\theta}) - C^*),$$

for some $\mu > 0$ and the global minimum C^*.

This says: the gradient is at least *some* fraction of the cost gap. Equivalently, you can't have a small gradient with a large cost gap.

Under PL: - SGD converges at the *deterministic* rate, not the slower stochastic rate (you can think of PL as saying "the noise floor doesn't matter as long as you're far from the minimum"). - The rate is $O(\rho^k)$ (linear) for the cost, not just for the gradient.

Does PL hold for VQAs? - Globally: very rarely. - Locally near a minimum: often, with μ related to the local Hessian eigenvalues. - In the barren plateau regime: definitely not (small gradient, large cost gap).

So PL is a **regional** property that you can sometimes invoke for late-stage convergence proofs.

SGD Convergence Under L-Smooth Non-Convex

The cleanest result that applies to VQAs:

Theorem (smooth non-convex SGD): For L-smooth C with bounded gradient variance σ^2, the iterates of stochastic GD with step size $\eta = c/\sqrt{k}$ satisfy

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|\nabla C(\boldsymbol{\theta}_k)\|^2] \leq \frac{C(\boldsymbol{\theta}_0) - C^*}{c\sqrt{K}} + \frac{cL\sigma^2}{\sqrt{K}}.$$

In words: the *average squared gradient norm* over the trajectory is $O(1/\sqrt{K})$.

Implications:

- Average gradient eventually decreases — we are converging toward a critical point.
- The rate $O(1/\sqrt{K})$ is slow but nontrivial.
- Doubling K only reduces the bound by $\sqrt{2}$, not 2. So 4x more iterations halves the bound.
- The constant depends on σ^2 — the *gradient variance* — which for VQAs is set by shot noise plus barren plateau effects.

For shallow VQAs without barren plateau, this gives a useful (if loose) guarantee. For deep VQAs in the plateau, σ^2 explodes and the bound is vacuous.

Convergence Rates For Different Optimizers

A summary table for the optimizers covered in Module 6:

Optimizer	Rate (cost)	Rate (grad norm)	Cost per iteration	Where it applies
Vanilla GD (deterministic)	$O(1/k)$	$O(1/k)$	$2P$	Smooth convex
Vanilla SGD	$O(1/\sqrt{k})$	$O(1/\sqrt{k})$	$2P$	Smooth non-convex

Optimizer	Rate (cost)	Rate (grad norm)	Cost per iteration	Where it applies
Natural Gradient	$O(1/k)$ (better constants)	$O(1/k)$	$2P + O(P^2)$	Smooth convex
Adam	$O(1/\sqrt{k})$	$O(1/\sqrt{k})$	$2P$	Smooth non-convex
SPSA	$O(1/\sqrt{k})$	$O(1/\sqrt{k})$	2	Smooth non-convex
Population (CMA-ES)	Empirically $O(1/k)$	N/A (zero-order)	O(pop size)	Multi-modal

The **rates** are roughly the same across optimizers (for non-convex problems, everyone gets $O(1/\sqrt{k})$ at best). The differences are in:

1. **Constants** (Natural Gradient and Adam have better constants in practice).
2. **Per-iteration cost** (SPSA has $O(1)$, Natural Gradient has $O(P^2)$).
3. **Robustness regime** (Adam handles ill-conditioning, SPSA handles many parameters cheaply).

The **total cost** to reach a fixed accuracy is a product of (rate) and (per-iteration cost), and the best optimizer depends on which factor dominates.

What The Theory Doesn't Cover

Things that VQA convergence theory does *not* address well:

1. Barren plateaus. No clean convergence rate when the gradient is below the noise floor. The theorems either give vacuous bounds or assume away the plateau.

2. Non-stationary noise. Hardware noise drifts over hours/days, but stochastic-optimization theory assumes i.i.d. noise. Drift breaks the analysis.

3. Biased gradients. Hardware noise can bias the gradient estimator (the noisy expected gradient differs from the true gradient). Standard theory assumes unbiased estimators.

4. Discrete decisions. Many VQA optimizers make discrete choices — which Pauli grouping to measure, which parameter to update next, etc. — that aren't covered by smooth-optimization theory.

5. Population dynamics. Population-based optimizers (HOPSO, CMA-ES) have their own convergence theories but they don't fit the SGD framework.

For these reasons, VQA optimizer practice usually relies on **empirical** convergence behavior with theory as a sanity check, not the other way around.

A Useful Theoretical Frame: Implicit Regularization

A more recent theoretical perspective: **the choice of optimizer implicitly regularizes the solution.**

In classical deep learning, it has been shown that SGD has an implicit bias toward "flat minima" (minima with small Hessian eigenvalues), which generalize better. This implicit bias is partly responsible for why SGD outperforms full-batch GD on overparameterized neural networks, even though both should converge to the same minima.

The same idea applies to VQAs: different optimizers find different minima, even when starting from the same initialization. Adam tends to find flatter minima than vanilla GD; natural gradient finds minima that are flat in the Fubini-Study metric; SPSA finds minima robust to per-coordinate perturbations.

This is the bridge to **regularization** (Module 5): the optimizer is not just a tool for finding the minimum — it's a mechanism for *selecting* among the minima a given cost landscape contains. This is the thesis's "dynamics-side regularization" view in another guise.

Stochastic Convergence Theorems For VQA-Specific Settings

A few results worth knowing that are tailored to VQA:

Sweke et al. (2020): "Stochastic gradient descent for hybrid quantum-classical optimization." Showed that even with single-shot gradient estimates ($N = 1$ per parameter-shift evaluation), SGD on VQAs converges in the smooth non-convex sense, provided the learning rate decays appropriately. This is theoretically reassuring — you don't need many shots per estimate, you can use single shots and rely on the optimizer to average.

Harrow and Napp (2021): "Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms." Showed that more efficient gradient measurements (jointly measuring multiple gradient components) can reduce variance and accelerate convergence.

Skolik et al. (2021): "Layerwise learning for quantum neural networks." Provided the theoretical justification for layerwise training as a way to avoid the barren plateau regime, with explicit convergence guarantees in the layerwise setting.

These are examples of theory tailored to VQA structure rather than borrowed wholesale from classical optimization.

Structural Role

This node is the **theoretical scaffolding** of Module 6. It provides the convergence rate framework that subsequent sections use, identifies the regimes where theory does and doesn't apply, and connects to the implicit-regularization view that bridges to Module 5.

Relations to Other Concepts

- **Robbins-Monro 1951** — foundational convergence theorem for stochastic approximation.
- **L-smoothness / Lipschitz gradients** — the standard structural assumption.
- **PL condition** — weakened convexity sufficient for fast rates.
- **Implicit regularization (deep learning)** — the optimizer-as-regularizer view.
- **Sweke et al. 2020** — single-shot SGD convergence for VQA.

Worked Example — Convergence Rate For A Smooth Convex VQE Toy

Consider $C(\theta) = (1 - \cos(\theta))^2/4$, a smooth convex cost on a single parameter with minimum at $\theta = 0$ and $C^* = 0$.

Compute the smoothness constant. $C'(\theta) = (1 - \cos\theta)\sin\theta/2$. $|C''(\theta)| \leq 1$ (verifiable by differentiating). So $L = 1$.

Vanilla GD with $\eta = 0.5$: the convergence rate for smooth convex is $O(1/k)$, so $C(\theta_k) - C^* \leq C(\theta_0)/(1+k\eta)$.

Starting from $\theta_0 = \pi/2$: $C(\pi/2) = 1/4$. Predicted bound at $k = 100$: $(1/4)/51 \approx 0.005$. The actual error after 100 steps is similar (the bound is loose but in the right ballpark).

SGD with shot noise $\sigma = 0.05$: the bound becomes $O(\sigma^2/\sqrt{k})$ instead of $O(1/k)$, because the noise floor dominates after enough iterations. At $k = 100$, the bound is $0.05^2/10 = 0.00025$ — actually *better* than the deterministic bound! This is because the $O(1/k)$ deterministic bound is loose, and the SGD bound only "pays" for the noise asymptotically.

In practice, deterministic GD reaches 10^{-4} precision in ~100 iterations. SGD with the same step size and $\sigma = 0.05$ reaches 10^{-3} precision in ~100 iterations and stagnates near the noise floor. To reach 10^{-4}, you'd need either much more shots (lower σ) or many more iterations.

This is the canonical "rate vs noise floor" tradeoff that shapes all stochastic optimization.

Visualization

Pending. A log-log plot of $C(\boldsymbol{\theta}_k) - C^*$ vs iteration k for several optimizers (deterministic GD, stochastic GD, Adam, SPSA, natural gradient). Each shows a different rate and a different noise floor. Caption: "Rates differ; noise floors differ; total budget differs."

Exercises

1. Verify that $C(\theta) = \cos(\theta)$ is L-smooth with $L = 1$. Compute the convergence rate of GD with $\eta = 1$ starting at $\theta_0 = 0.1$, targeting the maximum at $\theta = 0$. (Note: this is a maximum, not a minimum — but the same theorem applies.)
2. The PL condition is $\frac{1}{2}\|\nabla C\|^2 \geq \mu(C - C^*)$. For $C(\theta) = \cos(\theta) - \cos(\theta^*)$ near $\theta = \theta^*$, what is μ?
3. (VQA) For a 30-parameter HEA at a fixed shot budget of 10^6 total shots, estimate which optimizer (vanilla GD, Adam, SPSA, natural gradient) will give the lowest final cost. State your assumptions.

Research Extensions

- **VQA-specific PL conditions** — finding regions of VQA cost landscapes where PL holds and quantifying the constant.
- **Convergence under hardware noise drift** — extending stochastic optimization theory to non-stationary noise.
- **Convergence under biased gradients** — VQA with biased estimators.
- **Implicit regularization theory for VQA optimizers** — formalizing which optimizers prefer which kinds of minima.

Cross-links to Other Courses

- **Convex Optimization (Boyd-Vandenberghe)** — foundational convergence results.
- **Stochastic Approximation (Kushner-Yin)** — asymptotic theory of stochastic recursions.
- **Module 4 (Barren Plateaus)** — the regime where convergence theory fails to give useful bounds.
- **Module 5 (Regularization)** — the implicit regularization perspective on optimizer choice.

Optimal Transport and Optimization

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.6

This is the most **abstract** node in Module 6, and one of the most thesis-aligned.

The standard view of optimization is: pick a starting point in parameter space, follow a descent direction, end up at a local minimum. The **optimal transport** view is different: the optimizer is moving a *probability distribution* over parameter space, not a single point. The "trajectory" is a path of distributions, evolving according to a flow that decreases the expected cost.

This view is more general than the point-based view. It makes population-based optimizers (Section 6.7) look like a natural special case rather than an exotic alternative. It connects to the **gradient flow** view of differential equations, the **Wasserstein metric** on distributions, and the thesis's regularization framework.

This node introduces the optimal transport view of optimization, the technical machinery (Wasserstein gradient flow, JKO scheme), and explains why this perspective matters for VQA.

From Points To Distributions

In standard optimization, the optimizer's state is a single point $\boldsymbol{\theta}_k \in \mathbb{R}^P$.

In the distributional view, the optimizer's state is a probability distribution μ_k over $\mathbb{R}^P$. The state evolves over time according to a rule that depends on the cost function C.

A point-based optimizer is a special case: $\mu_k = \delta_{\boldsymbol{\theta}_k}$, a delta function at the current parameter. The distributional view collapses to the point view when the distribution has no spread.

A population-based optimizer is also a special case: $\mu_k = \frac{1}{N}\sum_{i=1}^{N} \delta_{\boldsymbol{\theta}_k^{(i)}}$, an empirical distribution from N particles. The distributional view captures the population's collective behavior.

A Bayesian optimizer is yet another case: μ_k is the posterior over parameters given the observed costs.

The distributional view unifies these.

Wasserstein Gradient Flow

The principal mathematical tool for distributional optimization is **Wasserstein gradient flow**.

The Wasserstein-2 distance between two distributions μ and ν is

$$W_2(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int \|x - y\|^2 \, d\pi(x, y) \right)^{1/2},$$

where $\Pi(\mu, \nu)$ is the set of joint distributions with marginals μ and ν. Intuitively, W_2 measures the "cost of transporting mass" from μ to ν, where the cost is squared Euclidean distance.

The space of distributions equipped with W_2 is a *metric space* (in fact, a Riemannian-like manifold called Wasserstein space).

A **Wasserstein gradient flow** is a curve μ_t in Wasserstein space that decreases a functional $\mathcal{F}[\mu]$ at each instant. The defining equation is

$$\partial_t \mu_t = -\nabla_{W_2} \mathcal{F}[\mu_t].$$

For $\mathcal{F}[\mu] = \int C(\boldsymbol{\theta})\, d\mu(\boldsymbol{\theta})$ (the expected cost under μ), the Wasserstein gradient flow is

$$\partial_t \mu_t = \nabla \cdot (\mu_t \nabla C),$$

which is a transport equation: each "particle" of mass moves with velocity $-\nabla C$, and the distribution as a whole flows downhill.

This is the continuous-time, distributional analogue of gradient descent.

Discretization: The JKO Scheme

To turn the continuous flow into an algorithm, you discretize in time. The canonical discretization is the **Jordan-Kinderlehrer-Otto (JKO) scheme**:

$$\mu_{k+1} = \arg\min_{\nu} \left[\mathcal{F}[\nu] + \frac{1}{2h} W_2^2(\nu, \mu_k) \right].$$

In words: move from μ_k to a new distribution ν that decreases $\mathcal{F}$ but doesn't move too far in Wasserstein distance.

This is a variational formulation: each step solves a small optimization problem (minimize cost plus transport penalty). The transport penalty acts as a regularizer that limits how much the distribution can change in one step — analogous to the trust-region constraint in classical optimization.

In the limit $h \to 0$, the JKO sequence converges to the continuous Wasserstein gradient flow.

For VQA: the JKO scheme can be implemented by an optimizer that maintains an empirical distribution (a population of points), updates it via cost-decreasing moves, and ensures the distribution doesn't change too much per step. Population methods like CMA-ES and particle swarm can be viewed as approximate JKO schemes.

Why This Matters For VQA

The optimal transport view brings several benefits to VQA optimization:

1. Natural framework for population methods. Population optimizers (HOPSO, CMA-ES, genetic algorithms) are point-based at the algorithm level but distributional at the conceptual level. The Wasserstein gradient flow gives them a clean theoretical foundation.

2. Better handling of multi-modal landscapes. A distributional optimizer can have mass at multiple local minima simultaneously, and the relative weights can shift over time as the optimizer learns which minimum is best. A point-based optimizer is committed to one location.

3. Natural connection to regularization. Many regularization terms (entropy regularization, trust regions, KL penalties) are naturally distributional. Adding them to the JKO objective gives a clean way to design **regularized population optimizers**, which is one of the threads of the thesis.

4. Plateau-aware exploration. A distributional optimizer with high entropy (broad distribution) explores plateau regions more efficiently than a point optimizer, because it samples multiple parameter values simultaneously and can discover gorges or non-plateau regions without making sequential decisions.

5. Connection to physics. Wasserstein gradient flow is the natural continuous-time limit of Brownian motion in a potential, which is a model from statistical physics. This gives a physical intuition for the optimizer dynamics — the optimizer is a "particle" moving in a "potential landscape" with "thermal noise."

Entropy Regularization And Exploration

A particularly important variant: **entropy-regularized JKO**.

Modify the JKO objective to include an entropy term:

$$\mu_{k+1} = \arg\min_{\nu} \left[\int C \, d\nu - \tau H[\nu] + \frac{1}{2h} W_2^2(\nu, \mu_k) \right],$$

where $H[\nu] = -\int \nu \log \nu \, d\nu$ is the differential entropy and $\tau > 0$ is a temperature parameter.

The continuous-time limit is the **Fokker-Planck equation**:

$$\partial_t \mu_t = \nabla \cdot (\mu_t \nabla C) + \tau \Delta \mu_t.$$

This is the equation of a Brownian particle in the potential C at temperature τ. The optimizer **explores the landscape** with thermal-like fluctuations.

Properties:

- High τ: broad distribution, lots of exploration, slow descent.
- Low τ: narrow distribution, fast descent, prone to local minima.
- $\tau \to 0$: recovers the deterministic Wasserstein gradient flow.

Annealing τ over time (start high, decrease) is analogous to **simulated annealing**, and gives a principled exploration-exploitation schedule.

For VQAs, entropy-regularized population optimizers can escape barren plateaus by maintaining enough diversity (high entropy) to keep some particles outside the plateau region, where they continue to make progress and pull the rest of the population along.

Sinkhorn And Practical Wasserstein

In practice, computing Wasserstein distances exactly is expensive. The standard practical tool is **Sinkhorn divergence** (Cuturi 2013), which is an entropy-regularized approximation that can be computed in $O(n^2)$ for n particles via fixed-point iteration.

Sinkhorn-based Wasserstein gradient flows have become a workhorse in machine learning (generative models, domain adaptation, etc.) and are starting to appear in VQA as well.

For population-based VQA optimizers, Sinkhorn-based updates allow you to maintain the distributional view computationally.

The Thesis Connection

The thesis argues that **regularization should be viewed as dynamical modification rather than objective modification**. The Wasserstein gradient flow framework makes this explicit:

- **Objective-side regularization:** modify $\mathcal{F}[\mu] = \int C \, d\mu$ by adding a penalty term, e.g., $\mathcal{F}[\mu] = \int C \, d\mu + \lambda R[\mu]$.
- **Dynamics-side regularization:** modify the *flow equation* directly, e.g., add diffusion: $\partial_t \mu = \nabla \cdot (\mu \nabla C) + \tau \Delta \mu$.

These are *not* the same. Adding entropy penalty to the objective is not the same as adding diffusion to the flow — they have different stationary distributions and different transient behavior. The thesis dwells on this distinction at length, and Module 6 (this node specifically) is where it gets the cleanest statement.

The optimal transport view is **the natural language** in which to make this distinction precise.

Limitations And Practical Considerations

The Wasserstein view is conceptually beautiful but has practical limitations:

1. Cost. Maintaining a distribution requires more state (a population of particles, or a density estimate) than maintaining a single point. Each iteration is more expensive.

2. Convergence rates. Wasserstein gradient flow has its own convergence theory, but the rates are typically slower than point-based optimizers in the deterministic regime. The benefit shows up in non-convex / multi-modal / noisy settings, not in clean convex problems.

3. Discretization artifacts. JKO discretization introduces its own errors, and the Sinkhorn approximation introduces more. The distributional view is mathematically clean but algorithmically messier than point-based methods.

4. Limited adoption. Most VQA software packages don't have first-class support for distributional optimizers. You'd typically implement them by hand or use a population optimizer (Section 6.7-6.8) as the practical proxy.

For these reasons, optimal-transport-based VQA optimization is more of a theoretical viewpoint than a practical tool in 2026. But it's an important viewpoint, because it grounds population methods and dynamics-side regularization in a coherent framework.

Structural Role

This node is the **distributional viewpoint** of Module 6. It abstracts the point-based optimizers (6.1-6.5) into a distributional framework, motivates population methods (6.7-6.8) as natural special cases, and provides the cleanest setting for the thesis's distinction between objective-side and dynamics-side regularization.

Relations to Other Concepts

- **Wasserstein metric** — the distance on the space of distributions.
- **JKO scheme (Jordan-Kinderlehrer-Otto 1998)** — the variational discretization.
- **Fokker-Planck equation** — the entropy-regularized continuous flow.
- **Sinkhorn algorithm (Cuturi 2013)** — practical Wasserstein computation.
- **Simulated annealing** — the analogue of entropy-regularized flow with annealed temperature.

Worked Example — Wasserstein Gradient Flow For A 1D Cost

Consider $C(\theta) = \cos(\theta)$ on $[-\pi, \pi]$, with global minimum at $\theta = \pi$ (or equivalently $-\pi$).

A point optimizer starting at $\theta_0 = 0.5$ follows GD: $\theta_{k+1} = \theta_k - \eta \cdot (-\sin(\theta_k))$, reaching the minimum.

A distributional optimizer starting at $\mu_0 = \mathcal{N}(0.5, 0.5^2)$ (a Gaussian centered at 0.5 with std 0.5) follows the JKO scheme. Each iteration: - Compute the expected cost $\int C \, d\mu_k$ and its gradient. - Move the distribution in the descent direction, with a Wasserstein penalty limiting the step.

The distribution slowly drifts toward $\theta = \pi$, simultaneously narrowing as the entropy decreases (in unregularized JKO) or staying broad (in entropy-regularized JKO).

After many steps, the unregularized JKO converges to a delta at π, while entropy-regularized JKO converges to a Boltzmann distribution $\exp(-C/\tau)$ concentrated near π but with some spread.

For a multi-modal landscape (e.g., $C(\theta) = \cos(2\theta)$ with minima at $\pi/2$ and $-\pi/2$), the distributional optimizer can have mass at both minima simultaneously, while the point optimizer is committed to one.

This is the conceptual benefit: distributional optimizers handle multi-modal landscapes natively.

Visualization

Pending. A 1D plot showing the evolution of μ_t over time on a non-convex $C(\theta)$ landscape. Initial Gaussian, then progressively concentrated at the global minimum. Side-by-side with a point-based GD trajectory. Caption: "The distribution flows downhill. The point only moves once."

Exercises

1. For $C(\theta) = \theta^2$ and $\mu_0 = \mathcal{N}(2, 1)$, compute the JKO update with $h = 0.1$. What is μ_1?
2. The Fokker-Planck equation has stationary distribution $\mu^* \propto \exp(-C/\tau)$. Verify this for $C(\theta) = \theta^2$. What is the limit as $\tau \to 0$?
3. (VQA) Sketch how an entropy-regularized population optimizer would behave on a 2D barren plateau landscape with a small "gorge" containing the global minimum. Compare to a point-based optimizer with random init.

Research Extensions

- **Sinkhorn-based VQA optimizers** — practical population optimizers with Wasserstein structure.
- **Wasserstein natural gradient** — combining the Fubini-Study metric with the Wasserstein metric.
- **Entropy regularization for barren plateaus** — using temperature annealing to escape plateaus.
- **JKO schemes for VQA** — algorithmic implementations of the JKO discretization tailored to quantum costs.

Cross-links to Other Courses

- **Optimal Transport (Villani, Peyré-Cuturi)** — the underlying mathematical theory.
- **Statistical Mechanics** — Brownian motion, Fokker-Planck, thermal exploration.
- **Generative Models in ML** — Wasserstein GANs and related distributional methods.
- **Module 5 (Regularization)** — the dynamics-side framework.
- **Module 6 Section 6.7 (Population-Based Optimizers)** — the practical embodiment.

Population-Based Optimizers

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.7

A point-based optimizer (everything in 6.1-6.5) maintains a single parameter vector and updates it based on local information. A **population-based optimizer** maintains a *set* of parameter vectors — a population — and updates them collectively.

This is a different optimization paradigm. Instead of "follow the gradient," it's "explore via diversity." Instead of "find a critical point," it's "concentrate the population on good regions of parameter space."

Population methods are the natural fit for VQA in two specific situations:

1. **Multi-modal landscapes** where a single optimizer would get stuck in a local minimum.
2. **Plateau regions** where individual optimizers can't make progress, but a population can collectively discover non-plateau regions.

This node introduces the population-based framework, walks through the major algorithm families (genetic algorithms, evolution strategies, particle swarm, CMA-ES), and sets up node 6.8 (HOPSO, the thesis's preferred hybrid).

Why Populations?

The basic argument for populations:

1. Implicit parallelism. A population of N particles samples N different parameter regions simultaneously. Even if any single particle gets stuck, the population as a whole has more information about the landscape.

2. Diversity preservation. Population methods can be designed to maintain diversity (spread among particles), which prevents premature convergence to a single minimum. This is the population analogue of entropy regularization.

3. No gradient required. Most population methods are zero-order — they only need cost evaluations, no gradients. This makes them robust to non-differentiable cost functions, discrete parameters, and simulation-based settings.

4. Robust to noise. Aggregating over a population reduces the influence of noise on any individual evaluation. The population's mean (or median) is a more robust statistic than any single particle's cost.

5. Embarrassingly parallel. The N cost evaluations per iteration are independent and can be run in parallel on N quantum devices (or sequentially with no synchronization).

The cost: each iteration evaluates N costs instead of 1, so per-iteration shot budget is N times higher. The benefit needs to outweigh this cost.

The Algorithm Families

Population methods come in several distinct families:

1. Genetic Algorithms (GA)

- Maintain a population of "individuals" (parameter vectors).
- Each generation: evaluate fitness, select the best, produce offspring via mutation and crossover.
- Inspired by biological evolution.

Strengths: very general, work on discrete and combinatorial problems. **Weaknesses:** slow, lots of hyperparameters, hard to tune.

2. Evolution Strategies (ES)

- Similar to GA but use Gaussian mutations (no crossover).
- The current "best" is updated by averaging successful mutations.
- More principled than GA, with cleaner convergence properties.

Strengths: simpler than GA, scales better, has theoretical foundations. **Weaknesses:** still slow compared to gradient methods on smooth problems.

3. CMA-ES (Covariance Matrix Adaptation Evolution Strategy)

- Hansen and Ostermeier (2001).
- The current "gold standard" of evolution strategies.
- Maintains a *covariance matrix* over parameters, adapted online to capture the local landscape geometry.
- Effectively learns a natural-gradient-like preconditioner from the population statistics.

Strengths: very robust, handles ill-conditioning, often competitive with gradient methods on noisy problems. **Weaknesses:** $O(P^2)$ covariance matrix, expensive for large P.

4. Particle Swarm Optimization (PSO)

- Eberhart and Kennedy (1995).
- Particles move through parameter space with velocities updated based on:
  - Inertia (continue current direction).
  - Personal best (return to best point seen).
  - Global best (move toward best point in population).
- Inspired by flocking behavior of birds.

Strengths: simple, intuitive, easy to implement. **Weaknesses:** can prematurely converge, requires careful inertia tuning.

5. Differential Evolution (DE)

- Storn and Price (1997).
- Update each particle by adding the difference between two random other particles.
- Simple and effective on rugged landscapes.

Strengths: simple, robust, few hyperparameters. **Weaknesses:** slow on smooth problems, no built-in noise handling.

6. Bayesian Optimization (BO)

- Maintain a Gaussian process posterior over the cost function.
- At each iteration, query the point that maximizes an acquisition function (e.g., expected improvement).
- Population is implicit (the GP uses all past observations).

Strengths: very data-efficient, principled handling of uncertainty. **Weaknesses:** $O(k^3)$ cost in number of past observations, doesn't scale to high P.

For VQA, **CMA-ES** and **PSO** (and the HOPSO variant in 6.8) are the most commonly used population methods. **BO** is used when shots are extremely expensive (real hardware experiments with limited time).

CMA-ES In Detail

CMA-ES is the canonical sophisticated evolution strategy. Here's the core update.

State: a mean vector $\mathbf{m}_k \in \mathbb{R}^P$, a covariance matrix $C_k \in \mathbb{R}^{P\times P}$, and a step size $\sigma_k > 0$.

At each generation:

1. **Sample.** Draw λ candidate solutions $\boldsymbol{\theta}_k^{(i)} \sim \mathcal{N}(\mathbf{m}_k, \sigma_k^2 C_k)$ for $i = 1, \ldots, \lambda$.
2. **Evaluate.** Compute $C(\boldsymbol{\theta}_k^{(i)})$ for each candidate.
3. **Select.** Sort by cost and keep the top $\mu < \lambda$ (the "elite").
4. **Update mean.** $\mathbf{m}_{k+1} = \sum_{i=1}^{\mu} w_i \boldsymbol{\theta}_k^{(i)}$, a weighted average of the elite.
5. **Update covariance.** Adapt C_k using the elite's distribution: increase variance in directions where the elite spread out, decrease in directions where it concentrated.
6. **Update step size.** Adapt σ_k based on the recent successful directions.

The covariance adaptation is the magic: over many generations, C_k converges to (a scalar multiple of) the local Hessian inverse, so the sampling distribution becomes aligned with the landscape geometry. CMA-ES is, in this sense, a population-based approximation to natural gradient — and it doesn't need any gradient computations.

For a detailed treatment, see Hansen's tutorial (2016).

CMA-ES For VQA

CMA-ES has been used for VQA in several papers, with mixed results:

When it works: - $P < 50$ (so the covariance matrix is manageable). - The cost landscape is smooth and the local structure is approximately quadratic. - Shot noise is moderate (CMA-ES handles it via population averaging).

When it doesn't: - Large P (covariance matrix becomes prohibitive). - Strong barren plateaus (CMA-ES gets confused when there's no signal). - Very rugged landscapes (the covariance adaptation lags behind the local structure).

For typical VQE problems with 10-30 parameters, CMA-ES is a competitive alternative to Adam, sometimes finding lower minima at the cost of more total cost evaluations. For larger problems, it's typically infeasible.

Particle Swarm For VQA

PSO is simpler and scales better but has its own issues:

Update rule for particle i:

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1(\mathbf{p}_i - \boldsymbol{\theta}_i^k) + c_2 r_2(\mathbf{g} - \boldsymbol{\theta}_i^k),$$
$$\boldsymbol{\theta}_i^{k+1} = \boldsymbol{\theta}_i^k + \mathbf{v}_i^{k+1},$$

where w is the inertia, c_1, c_2 are acceleration coefficients, r_1, r_2 are uniform random numbers, $\mathbf{p}_i$ is particle i's personal best, and $\mathbf{g}$ is the global best.

Strengths: - Very simple, no covariance matrix. - Scales linearly in P. - Easy to interpret and modify.

Weaknesses: - Can prematurely converge to the global best, losing diversity. - Sensitive to inertia/acceleration tuning. - Doesn't adapt to landscape geometry like CMA-ES does.

For VQA with many parameters, PSO is one of the few population methods that remains tractable. It is the foundation for HOPSO (Section 6.8), which is the thesis's specific PSO variant.

The Diversity Problem

A central tension in population methods: **how to maintain diversity without losing convergence speed**.

Too much diversity: the population explores uniformly but never converges. Too little diversity: the population collapses to a single point (effectively becoming a point optimizer) and loses the benefits of being a population.

Mechanisms to maintain diversity:

- **Niching:** explicitly assign different particles to different "niches" of parameter space.
- **Crowding:** penalize particles that get too close to each other.
- **Mutation rate:** keep mutation high enough to spread the population.
- **Restart:** periodically reset some particles to random points.
- **Multi-modal selection:** in CMA-ES, use multi-modal variants that maintain multiple covariance matrices.

For VQA, the right diversity strategy depends on whether the landscape is multi-modal (use diversity-preserving variants) or unimodal but plateau-prone (use restart-based variants).

Population Methods And Stochastic Optimization Theory

Population methods don't fit the classical stochastic optimization framework cleanly. The convergence theory for ES, CMA-ES, and PSO is its own subfield, with weaker guarantees than SGD theory.

That said, several recent results have shown that **CMA-ES is approximately equivalent to stochastic natural gradient on a Gaussian distribution** (Akimoto et al., Ollivier et al.), and that **PSO is approximately equivalent to a discrete-time approximation of the Wasserstein gradient flow** (Section 6.6).

This gives a theoretical foundation that connects population methods to the rest of the optimization literature, even though they look quite different at the algorithm level.

Structural Role

This node is the **distributional / population** entry point of Module 6. It introduces the population-based paradigm, surveys the major algorithms, and sets up node 6.8 (HOPSO) as the specific hybrid that the thesis advocates.

It also forwards directly to Module 5 (regularization), where many regularization techniques have natural population-based interpretations.

Relations to Other Concepts

- **Hansen 2016 (CMA-ES tutorial)** — the canonical reference for CMA-ES.
- **Eberhart-Kennedy 1995 (PSO)** — the original particle swarm paper.
- **Ollivier et al. 2017** — connecting CMA-ES to natural gradient.
- **Wasserstein gradient flow (6.6)** — the continuous limit of population dynamics.
- **Module 5 regularization** — population methods naturally implement many regularization ideas.

Worked Example — PSO On A 2D Cost

Consider $C(\theta_1, \theta_2) = (\theta_1 - 1)^2 + (\theta_2 - 2)^2$, with the minimum at $(1, 2)$.

Setup: 5 particles, initialized uniformly at random in $[-3, 3]^2$. Inertia $w = 0.7$, acceleration $c_1 = c_2 = 1.5$, velocities initialized to zero.

Iteration 1: - Particle positions: $(1.5, -2), (-2, 1), (0, 0), (3, 3), (-1, -3)$. - Costs: $16.25, 10, 5, 5, 29$. - Global best so far: $(0, 0)$ or $(3, 3)$, tie at cost 5. Pick $(0, 0)$. - Each particle's personal best is its current position.

Iteration 2: - Each particle accelerates toward $(0, 0)$ (global best) and toward its personal best (which is its current position, so no contribution). - New velocities: $w \cdot 0 + c_1 r_1 \cdot 0 + c_2 r_2(\mathbf{g} - \boldsymbol{\theta}_i)$ — random pull toward the global best. - Particles move toward $(0, 0)$, but with noise.

Iteration 3-10: - The particles cluster around the current global best, but explore the area. - One eventually finds $(1, 2)$ (the true minimum) and becomes the new global best. - The rest of the population follows.

Convergence: typically ~ 30 iterations for 2D problems, ~ 200 for 10D problems, $\sim 1000+$ for higher dimensions. The cost per iteration is 5 evaluations (for 5 particles), so the total cost is 5x the number of iterations.

For VQA, this scales linearly in the number of particles (which gives the population effect) and roughly as a polynomial in P (PSO doesn't suffer from the curse of dimensionality as badly as some methods).

Visualization

Pending. A 2D contour plot of a multi-modal landscape with 5 PSO particle trajectories overlaid. Initial random scatter, then gradual convergence on the global minimum, with some particles briefly visiting local minima. Caption: "Particles explore. Eventually, they vote with their feet."

Exercises

1. For PSO with 3 particles on $C(\theta) = \theta^2$, starting at $\theta = -2, 0, 1$ with zero velocities, run 5 iterations by hand and verify convergence toward $\theta = 0$.
2. CMA-ES adapts a covariance matrix. For a quadratic cost $C = \boldsymbol{\theta}^T A \boldsymbol{\theta}$, what does the converged covariance matrix look like in terms of A?
3. (VQA) For a 100-parameter VQA, estimate the per-iteration shot budget for: (a) a vanilla parameter-shift gradient; (b) CMA-ES with 50 particles; (c) PSO with 50 particles. Which is most expensive?

Research Extensions

- **Quantum-aware population methods** — variants that exploit VQA structure (periodicity, parameter-shift exact gradients).
- **Hybrid gradient + population methods** — using parameter-shift gradients to inform PSO or ES updates.
- **Diversity-aware CMA-ES for plateau escape** — variants that maintain multi-modal populations on barren plateau landscapes.
- **Bayesian optimization with quantum priors** — incorporating quantum-specific structure into the GP prior.

Cross-links to Other Courses

- **Evolutionary Computation** — the broader field of bio-inspired optimization.
- **Bayesian Optimization** — Gaussian process optimization, expected improvement.
- **Module 6 Section 6.6 (Optimal Transport)** — the continuous-limit framework.
- **Module 6 Section 6.8 (HOPSO)** — the specific PSO variant.

HOPSO: Hybrid Particle-Swarm Optimizer

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.8

This is the **closing node** of Module 6 and the most thesis-specific.

HOPSO — **Hybrid Optimizer Particle Swarm Optimization** — is the optimizer the thesis uses as its primary case study for the regularization-as-dynamical-modification framework. It is a hybrid of:

1. **Population-based exploration** (from PSO, Section 6.7), giving it the ability to escape local minima and barren plateau regions through population diversity.
2. **Per-particle gradient descent** (from parameter-shift, Section 6.1), giving each particle the ability to refine its position via gradient information rather than relying purely on swarm dynamics.

The result is an optimizer that combines the global-exploration benefit of populations with the local-precision benefit of gradients. It is the thesis's preferred answer to "what optimizer should you use for a difficult VQA," with the understanding that "difficult" means "barren-plateau-prone, multi-modal, or both."

This node walks through HOPSO's design, why each component is justified, and how it embodies the regularization-as-dynamics-modification thesis.

The HOPSO Update Rule

Each particle in the swarm has a position $\boldsymbol{\theta}_i^k$ and a velocity $\mathbf{v}_i^k$. At each iteration:

Step 1: Local gradient. Each particle computes its local cost gradient $\widehat{\nabla C}(\boldsymbol{\theta}_i^k)$ via parameter-shift, with whatever shot budget is allocated.

Step 2: Velocity update. The velocity is updated by combining gradient descent with PSO swarm terms:

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_g(-\widehat{\nabla C}(\boldsymbol{\theta}_i^k)) + c_p r_p(\mathbf{p}_i - \boldsymbol{\theta}_i^k) + c_s r_s(\mathbf{g} - \boldsymbol{\theta}_i^k),$$

where: - w is the inertia (typically 0.5-0.9). - c_g is the gradient coefficient (the "gradient pull"). - c_p is the personal-best coefficient. - c_s is the swarm-best (global) coefficient. - r_p, r_s are uniform random numbers in $[0, 1]$. - $\mathbf{p}_i$ is particle i's personal-best position so far. - $\mathbf{g}$ is the swarm's global-best position so far.

Step 3: Position update. $\boldsymbol{\theta}_i^{k+1} = \boldsymbol{\theta}_i^k + \mathbf{v}_i^{k+1}$.

Step 4: Personal/global best update. If the new position has a better cost than $\mathbf{p}_i$, update $\mathbf{p}_i$. If better than $\mathbf{g}$, update $\mathbf{g}$.

Step 5: Diversity check. If the swarm has collapsed (all particles within some radius), inject mutation or restart some particles.

What Each Term Does

The four velocity terms each play a distinct role.

1. Inertia term $w\mathbf{v}_i^k$. Maintains the previous velocity, smoothing the trajectory. Like momentum in classical SGD.

2. Gradient term $c_g(-\widehat{\nabla C})$. Pulls each particle in its local descent direction. This is the "particle does GD" component. In a region where the gradient is reliable (above noise floor, no plateau), this makes the particle converge quickly.

3. Personal-best term $c_p r_p(\mathbf{p}_i - \boldsymbol{\theta}_i^k)$. Pulls the particle back toward its own best historical position. This term is *self-correcting*: if a particle wanders into a bad region, it gets pulled back to its best known location.

4. Global-best term $c_s r_s(\mathbf{g} - \boldsymbol{\theta}_i^k)$. Pulls the particle toward the swarm's global best. This is the *social* component, which lets information flow between particles. The swarm collectively concentrates on the best region found so far.

The mix of gradient (local information) and PSO terms (population information) is what makes HOPSO a *hybrid*.

Why HOPSO Works In The Plateau Regime

The key claim of the thesis is that HOPSO can find solutions in regions where vanilla gradient descent fails because of barren plateaus.

Mechanism 1: Population diversity beats single-point starting. A vanilla GD optimizer with random initialization has a probability of landing in a non-plateau region that decays exponentially with qubit count. A swarm of N particles has a probability $\sim N \times 2^{-n}$ of having at least one particle land in a non-plateau region — still exponentially small, but multiplied by N. For $N = 100$, this is a 100x improvement, which can be the difference between "no signal" and "some signal."

Mechanism 2: Particles in the plateau follow the leader. Once one particle finds a non-plateau region (via the global-best pull, or by random PSO motion), the rest of the swarm is pulled toward it. The plateau-trapped particles don't need to find the signal themselves — they just need to follow the leader.

Mechanism 3: Gradient terms become useful again outside the plateau. Once a particle is outside the plateau, its local gradient is meaningful and the gradient term in HOPSO accelerates convergence. The PSO terms get the particle out of the plateau; the gradient term takes it from "in a non-plateau region" to "at a minimum."

Mechanism 4: Diversity preservation. HOPSO's diversity-check step prevents the swarm from collapsing entirely, which would destroy the population benefit. By keeping some spread, the optimizer continues to explore even after concentrating most of its mass.

This is the core thesis claim: **plateau escape via population diversity, plateau exploitation via local gradients**.

HOPSO And The Regularization-As-Dynamics Thesis

HOPSO's connection to the thesis runs deeper than "it works on plateaus."

The thesis distinguishes: - **Objective-side regularization:** modify the cost function (add penalty terms). - **Dynamics-side regularization:** modify the optimizer dynamics (don't touch the cost).

HOPSO is **purely dynamics-side**. The cost function is unchanged. The regularization happens through the optimizer's own structure: the population maintains diversity, the swarm dynamics encourage exploration, the gradient terms encourage exploitation. There is no added term in $C(\boldsymbol{\theta})$.

This is operationally meaningful because:

1. **You don't need to know the right penalty term.** Adding a penalty to C requires designing the penalty, choosing its weight, tuning its hyperparameters. HOPSO has its own hyperparameters (w, c_g, c_p, c_s) but they parameterize the optimizer's behavior, not the problem.
2. **You don't change the global minimum.** A penalty in C shifts the minimum (intentionally — that's its purpose). HOPSO leaves the minimum where it is. If the optimizer converges, it converges to the *original* minimum, not a regularized one.

3. **The regularization is adaptive.** The swarm's exploration vs exploitation balance shifts over time as the population concentrates. This is a natural form of annealing without requiring an explicit schedule.
4. **The framework is composable.** You can add objective-side regularization on top of HOPSO if you want both. The two are not mutually exclusive — they are different layers of the optimization stack.

The thesis argues that **dynamics-side regularization deserves equal billing with objective-side**, and HOPSO is a clean example of why: it solves real problems (barren plateau escape) without requiring any modification to the cost function.

Hyperparameters And Tuning

HOPSO has more hyperparameters than vanilla GD or even Adam. The main ones:

- N: population size. Typical $20 - 100$.
- w: inertia. Typical $0.5 - 0.9$.
- c_g: gradient coefficient. Typical $0.1 - 1.0$ (depends on landscape gradient scale).
- c_p: personal best coefficient. Typical $0.5 - 2.0$.
- c_s: global best coefficient. Typical $0.5 - 2.0$.
- Restart threshold: when the swarm has collapsed below this radius, inject mutations.
- Shot budget per particle per iteration.

This is more knobs than most VQA optimizers, which is a real cost. But the defaults work reasonably well across a wide range of problems, and tuning is rarely needed for casual use.

A practical default: $N = 50$, $w = 0.7$, $c_g = 0.3$, $c_p = 1.0$, $c_s = 1.0$, restart threshold 0.1, shot budget per particle = same as vanilla GD baseline.

When HOPSO Is And Isn't The Right Choice

HOPSO is a good choice for:

- **Difficult VQAs with barren plateaus** that vanilla GD can't escape.
- **Multi-modal landscapes** where the optimizer needs to explore multiple basins.
- **Noisy hardware** where the population averaging helps stabilize the trajectory.
- **Problems with adequate compute** to support a large population per iteration.

HOPSO is **not** a good choice for:

- **Simple smooth VQAs** where vanilla GD or Adam works fine. The population overhead is wasted.
- **Very high-dimensional problems** ($P > 200$) where the population can't densely sample parameter space anyway.
- **Time-critical experiments** where the per-iteration cost is prohibitive.
- **Problems with very expensive cost evaluations** (e.g., real hardware with limited time slots) where shooting your shot budget across 50 particles is too costly.

For "small-easy" problems, use Adam. For "big-hard" problems, use HOPSO. For everything in between, it depends on your shot budget and your patience.

Empirical Performance Of HOPSO

The thesis reports empirical comparisons of HOPSO against vanilla GD, Adam, natural gradient, and other baselines on a suite of VQA problems. The summary findings (paraphrased):

1. **On smooth, plateau-free problems** ($n \leq 8$ qubits, shallow ansatz), HOPSO is comparable to or slightly slower than Adam in total cost. The population overhead is wasted on easy problems.
2. **On medium-difficulty problems** ($n = 10 - 15$, moderate plateau risk), HOPSO is competitive with Adam and significantly more reliable — it succeeds on a higher fraction of random initializations.
3. **On hard problems** ($n = 18 - 24$, strong plateau risk, multi-modal), HOPSO finds significantly better minima than Adam or natural gradient. The gap widens with problem difficulty.
4. **On very hard problems** ($n > 24$ with deep ansatz), all optimizers struggle, but HOPSO degrades more gracefully than gradient methods.

The thesis interprets this as evidence that **population-based exploration becomes more important as problem difficulty grows**, which is exactly what the regularization-as-dynamics framing predicts.

Closing Module 6

Module 6 has now traversed the full optimizer landscape:

- **Vanilla GD (6.1):** the baseline.
- **Natural Gradient (6.2):** geometric correctness at high cost.
- **Adam (6.3):** practical adaptive methods, the workhorse default.
- **Stochastic Optimization (6.4):** the foundational theory.
- **Convergence Theory (6.5):** what we can prove.
- **Optimal Transport (6.6):** the distributional view.
- **Population Methods (6.7):** the practical population framework.
- **HOPSO (6.8, this node):** the thesis's hybrid recommendation.

The arc of the module is from "simplest possible optimizer" to "most sophisticated thesis-aligned hybrid." Each step adds a layer that addresses a failure mode of the previous one.

The most important take-away: **there is no single best optimizer for VQA**. The right choice depends on the problem's difficulty, the noise level, the parameter count, and the available compute. The thesis's specific recommendation — HOPSO for difficult problems, Adam for easy ones — is one reasonable rule, but the broader framework (point vs population, gradient vs zero-order, objective-side vs dynamics-side regularization) is what matters.

Module 7 will look at all of this through a control-theoretic lens. Module 8 will use it to build the expressivity-trainability tradeoff. Modules 9 and 10 will tie it back to hardware and deployment.

Structural Role

This node is the **operational closure** of Module 6. It is also the most direct embodiment of the thesis's central optimization-side argument: that population-based, dynamics-side regularization is a viable and underexploited alternative to objective-side fixes.

Relations to Other Concepts

- **PSO (Eberhart-Kennedy 1995)** — the parent algorithm.
- **CMA-ES** — alternative population method with adaptive geometry.
- **Hybrid optimization** — the broader category of gradient + population mixes.
- **Module 5 regularization** — the framework HOPSO embodies.
- **Wasserstein gradient flow (6.6)** — the continuous limit of HOPSO's swarm dynamics.

Worked Example — HOPSO On A Plateau-Prone Toy

Consider a 6-qubit HEA with 3 layers and a global cost $C = \langle Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 \rangle$.

Vanilla GD with random init. Initial gradient is ~ 0.05, shot noise floor is ~ 0.03. The optimizer barely moves, and after 1000 iterations, the cost has decreased by < 0.01. **Failure.**

Adam with random init. Same regime. Adam's adaptive step size doesn't help — the gradient is too noisy. Same failure.

HOPSO with $N = 30$ particles. - Iteration 1: 30 particles initialized randomly. Global best particle has cost ~ 0.1 (better than typical). - Iteration 10: the swarm has concentrated around the global best. A few particles have wandered into less-plateau regions and discovered cost ~ -0.2. - Iteration 50: the global best is now -0.6. The gradient at this point is meaningful (~ 0.3), and the gradient terms in HOPSO start contributing significantly. - Iteration 200: the swarm has converged on the global minimum at cost -1.

Total cost comparison: - HOPSO: 200 iterations $\times 30$ particles $\times (2P + 1)$ cost evaluations per iteration $=\sim 200 \times 30 \times 37 \approx 220{,}000$ cost evaluations. - Vanilla GD that fails: 1000 iterations $\times (2P + 1) \approx 37{,}000$ cost evaluations, but doesn't reach the answer.

HOPSO uses 6x more cost evaluations but actually finds the minimum. Vanilla GD uses fewer but fails.

This is the canonical HOPSO benefit: **more total cost, but finds answers that gradient methods cannot.**

Visualization

Pending. A 2D plot showing two trajectories on a multi-modal cost landscape with a plateau region: vanilla GD (gets stuck in the plateau, never reaches the minimum) and HOPSO (population spreads, one particle finds the minimum, rest of the swarm converges). Caption: "HOPSO escapes plateaus that GD cannot."

Exercises

1. For a 4-parameter ansatz with cost $C = \theta_1^2 + \theta_2^2 + \theta_3^2 + \theta_4^2$, run HOPSO with 5 particles for 10 iterations from random initialization. How fast does the global best converge to zero?
2. For HOPSO on a 1D bimodal cost $C(\theta) = \cos(2\theta)$, starting with a uniform initial swarm, sketch how the population evolves and which minimum it converges to.
3. (VQA) For a 20-qubit barren-plateau-prone HEA, estimate the minimum population size N such that HOPSO has $> 50\%$ probability of having at least one particle outside the plateau region at initialization. (Use any reasonable assumption about the plateau coverage fraction.)

Research Extensions

- **Adaptive HOPSO hyperparameters** — auto-tuning w, c_g, c_p, c_s based on swarm behavior.
- **HOPSO with quantum-aware mutations** — using parameter-shift periodicity to design mutation operators.
- **Multi-population HOPSO** — running multiple independent swarms in parallel, with occasional cross-population exchange.
- **HOPSO + natural gradient** — combining the swarm dynamics with QFIM-rescaled gradient terms.

Cross-links to Other Courses

- **Evolutionary Computation** — the parent field of population methods.
- **Module 5 (Regularization)** — the theoretical framework HOPSO embodies.
- **Module 6 Section 6.7 (Population Methods)** — the broader class.
- **Module 7 (Control-Theoretic View)** — alternative framing of the same dynamics.

Module 7 — Control Theoretic View

VQA as a Control System

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Control-Theoretic View **Node:** 7.1

This is the **opening node** of Module 7, the module that re-frames everything Modules 1-6 have built up — ansätze, costs, optimizers — in the language of **classical control theory**.

The motivation is more than aesthetic. Control theory has 70+ years of accumulated theory about steering dynamical systems with inputs, observing them with sensors, and designing feedback laws — *exactly* the same shape of problem a VQA poses. Importing the framework gives:

1. **A vocabulary** for talking about VQAs that connects to a mature engineering literature.
2. **Theorems** (controllability, observability, separation principle) that apply with minor modification.
3. **Design tools** (LQR, MPC, Pontryagin minimum principle) that can be applied to ansatz design and optimizer scheduling.
4. **A new perspective** on the open problems (barren plateaus, noise, regularization) that often illuminates them differently from the optimization-theoretic view.

The point of this module is *not* to replace the optimization view. It's to provide a second axis of understanding — one that is especially powerful for the questions Modules 9 (Hardware Noise) and 10 (VQA as Adaptive Control Systems) take up.

What A Control System Is

A **classical control system** is a dynamical system with:

- A **state** $\mathbf{x}(t) \in \mathcal{X}$ that evolves over time.
- A **control input** $\mathbf{u}(t) \in \mathcal{U}$ that the controller can choose.
- A **dynamics** $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$ that says how the state changes given the input.
- An **output** $\mathbf{y}(t) = h(\mathbf{x}, t) \in \mathcal{Y}$ that the controller can observe.
- An **objective** $J = \int L(\mathbf{x}, \mathbf{u})\, dt$ (or terminal cost) that the controller is trying to minimize.

The **control problem** is: choose $\mathbf{u}(t)$ so that J is minimized, possibly subject to constraints on $\mathbf{x}$, $\mathbf{u}$, and the observable measurements $\mathbf{y}$.

Different specializations give:

- **Open-loop control:** $\mathbf{u}(t)$ is chosen as a function of time alone, without observing the actual $\mathbf{x}(t)$.
- **Closed-loop / feedback control:** $\mathbf{u}(t) = K(\mathbf{y}(t), t)$ — the input depends on the measured output.
- **Optimal control:** the input is chosen to extremize a functional, typically using the Pontryagin minimum principle (Section 7.5).
- **Adaptive control:** the controller's parameters adapt over time in response to system behavior (Module 10).

A VQA Is A Control System

The mapping is direct:

Control concept	VQA analogue
State $\mathbf{x}$	Quantum state $\lvert\psi(\boldsymbol{\theta})\rangle$ (or its density matrix)
Control input $\mathbf{u}$	Circuit parameters $\boldsymbol{\theta}$ (or update steps)

Control concept	VQA analogue
Dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$	Schrödinger evolution under parameterized gates
Output $\mathbf{y}$	Measurement outcomes (estimated expectation values)
Objective J	Cost function $C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})\|H\|\psi(\boldsymbol{\theta})\rangle$
Open-loop	Pre-compiled, fixed-circuit VQA
Feedback	Parameter updates based on measurements
Optimal control	Choosing parameter trajectories to minimize cost
Adaptive control	Optimizers that change strategy based on observed behavior

This isn't an analogy — it's a translation. A VQA *is* a control system, with quantum dynamics replacing classical ODEs.

The question is what changes when you make this substitution. Most of control theory was developed for classical, deterministic, low-dimensional dynamical systems. Quantum systems are stochastic (measurement is intrinsically random), high-dimensional (2^n states), and have fundamentally different observability constraints (you collapse the state when you measure). So the import is not literal — it requires care.

Two Time Scales: The Inner And Outer Loops

A VQA has *two* dynamical loops, and the control framing makes them explicit.

Inner loop (state preparation). At fixed parameters $\boldsymbol{\theta}$, the quantum circuit evolves the state from $|0\rangle$ to $|\psi(\boldsymbol{\theta})\rangle$. The "time" variable here is the position in the circuit. Each gate is a control input that advances the state.

Outer loop (parameter optimization). The parameters $\boldsymbol{\theta}^k$ are updated iteration-by-iteration based on measurements of the cost. The "time" variable is the optimizer iteration. Each update is a control input that adjusts the *next* circuit's behavior.

These two loops have very different character:

- The **inner loop** is *unitary*, deterministic given $\boldsymbol{\theta}$, and runs in microseconds. Quantum control theory (which exists as a field) addresses it.
- The **outer loop** is *classical*, stochastic (because measurements are sampled), and runs in seconds-to-hours. This is where the optimization theory of Module 6 lives.

The control-theoretic view treats them as a *cascaded* control system: the outer loop controls the parameters, which in turn control the inner-loop dynamics, which in turn produce measurements, which feed back to the outer loop. Each level has its own controllability, observability, and stability properties.

What The Control Framing Buys You

A few specific things the control view gives that the pure-optimization view misses:

1. Controllability questions. Given a parameterized circuit, *which states can I steer the system to?* This is exactly the "reachability" question of expressivity (Module 8), but the control version connects it to a 60-year theorem stack about Lie algebras of vector fields, accessibility conditions, and the Chow-Rashevsky theorem.

2. Observability questions. Given measurement access to certain observables, *what can I learn about the underlying state?* This is the question of whether your cost function can resolve the parameter values

you care about. Classical observability theory has explicit conditions (the observability matrix rank) that translate cleanly.

3. The separation principle. Classical control theory says: under linearity and Gaussian noise, you can design the state estimator separately from the feedback law and the result is optimal. A quantum analogue exists in some regimes and gives a clean architecture for adaptive VQAs.

4. Robust control. What if the dynamics aren't exactly known? Classical robust control (H_∞, μ-synthesis) gives explicit guarantees on stability margins. The VQA analogue addresses noise and miscalibration, and gives quantitative robustness statements.

5. Model predictive control. Looking ahead a few steps and optimizing over the lookahead window. Naturally fits VQA optimizer design with lookahead — the optimizer "thinks" several iterations ahead.

The control framing is *especially* useful when noise, drift, or hardware time-variation are present — exactly the regimes where the static optimization view starts to break down.

Quantum Control Vs Classical Control

Quantum control has been a field in its own right since the 1980s. It addresses the inner-loop problem: given a quantum system with controllable Hamiltonian $H(\mathbf{u}(t))$, choose $\mathbf{u}(t)$ to steer the state from one configuration to another.

Major results:

- **Controllability theorems** (Schirmer-Solomon, D'Alessandro): a quantum system is fully controllable iff the Lie algebra generated by its drift and control Hamiltonians spans $\mathfrak{su}(2^n)$. Same Lie algebra story as Section 8.1 — the dynamical Lie algebra.
- **Optimal control algorithms** (GRAPE, Krotov, CRAB): iterative methods that find time-dependent control pulses to perform target unitaries. Used for high-fidelity gate calibration.
- **Decoherence-free subspaces and dynamical decoupling**: control protocols that protect quantum states from noise.

These results are *directly relevant* to VQAs. A parameterized circuit can be viewed as a discretization of an underlying continuous control problem. Many of the GRAPE-style insights transfer.

The thesis's view: **VQA optimization is quantum control with discrete inputs, and it inherits both the strengths and the limitations of the broader quantum control field.**

Where The Mapping Breaks Down

The control-theory analogy is powerful but imperfect. A few places it strains:

1. Discrete vs continuous time. Classical control is typically continuous time ($\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$) or discrete-time with regular step sizes. A VQA is event-driven: each gate is a discrete event with no fixed time step. Translating between continuous and discrete views introduces some friction.

2. Measurement is irreversible. Classical sensors don't disturb the state; quantum measurements collapse it. So "observe and feed back" is not free — it changes the system. Continuous quantum measurement theory (Wiseman-Milburn) addresses this but is more complex than classical observation theory.

3. The cost is itself stochastic. In classical control, you typically know J exactly. In a VQA, you only have a *noisy estimate* of the cost from finitely many shots. This is a stochastic-control problem, not a deterministic one.

4. The state space is exponentially large. $|\psi\rangle \in \mathbb{C}^{2^n}$. Classical control theory works for low-dimensional state spaces. The high dimension is what makes barren plateaus possible (and what makes quantum control hard in general).

5. Two-loop structure. Classical control is usually a single loop. The VQA two-loop structure (inner unitary loop, outer classical loop) requires extending the framework.

These caveats matter, and Module 7 will address them as we go. None is fatal, but each requires the standard control concepts to be modified before they apply cleanly.

The Module 7 Plan

Module 7 will develop the control-theoretic framework in stages:

- **Node 7.2 (Observability):** what can we learn from measurements? When does the cost function have enough resolution to identify the optimal parameters?
- **Node 7.3 (Controllability):** which states can we reach? The Lie-algebraic conditions for full and partial controllability.
- **Node 7.4 (Optimal Control Theory):** the variational framework for choosing trajectories. Connects to quantum optimal control (GRAPE).
- **Node 7.5 (Pontryagin Principle):** the necessary conditions for optimality. Gives a different angle on why VQA optimization is hard.
- **Node 7.6 (Feedback Control):** closing the loop. Adaptive optimizer design and the separation principle.

Module 7 is the bridge between Module 6 (optimization dynamics) and Module 10 (adaptive control systems). The theoretical tools live here; the deployment patterns live in 10.

Structural Role

This node is the **opening framing** of Module 7. It establishes the control-theoretic vocabulary, maps VQA concepts onto control concepts, identifies the two-loop structure, and previews the rest of the module.

It is also the bridge from the optimization-theoretic view (Modules 5, 6) to the adaptive deployment view (Modules 9, 10), with control theory as the connective tissue.

Relations to Other Concepts

- **Wiener (1948)** — cybernetics, the founding text of control as an interdisciplinary field.
- **Pontryagin et al. (1962)** — the minimum principle, the variational foundation of optimal control.
- **Schirmer-Solomon (2001)** — quantum controllability theorems.
- **D'Alessandro (2007)** — *Introduction to Quantum Control and Dynamics*, the textbook.
- **Module 6 (Optimization Dynamics)** — the optimization view, complementary to this one.

Worked Example — Mapping A QAOA Onto A Control System

Consider QAOA at depth $p = 2$ on a 4-qubit MaxCut problem. The circuit is

$$|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = e^{-i\beta_2 H_M} e^{-i\gamma_2 H_C} e^{-i\beta_1 H_M} e^{-i\gamma_1 H_C} |+\rangle^{\otimes 4},$$

where H_C is the cost Hamiltonian and H_M is the mixer.

Map onto control system.

- **State:** $|\psi(t)\rangle \in \mathcal{H} = \mathbb{C}^{16}$, evolving as the gates are applied.
- **Control input:** $\mathbf{u}(t)$ is piecewise constant, equal to β_1 on the first mixer interval, γ_1 on the first cost interval, etc. There are 4 distinct control intervals.
- **Dynamics:** Schrödinger equation $i|\dot{\psi}\rangle = H(\mathbf{u}(t))|\psi\rangle$, where $H(\mathbf{u})$ is either H_C or H_M depending on the interval.
- **Output:** measurement of H_C at the end, giving $y = \langle\psi(T)|H_C|\psi(T)\rangle$.
- **Objective:** minimize y (or equivalently maximize $-y$).

This is a **bang-bang control** problem — the control switches abruptly between two values H_C and H_M. The controller chooses how long to spend in each mode (the β_k, γ_k values) to minimize the terminal cost.

In control-theoretic language, QAOA is **discrete-time bang-bang optimal control with terminal cost**. The Pontryagin minimum principle (Section 7.5) gives explicit necessary conditions for the optimal β_k, γ_k, and these conditions match the gradient conditions from parameter-shift differentiation.

Why this matters: the mapping shows that QAOA is *not* an exotic quantum heuristic — it is a classical control problem in quantum disguise. The same theorems that govern bang-bang optimal control of classical systems govern QAOA, with appropriate modifications for the quantum state space.

Visualization

Pending. A block diagram of the VQA two-loop control system: inner loop (parameters → quantum circuit → state preparation → measurement → cost estimate), outer loop (cost estimate → optimizer → parameter update → back to inner loop). Caption: "VQA is a cascaded control system with quantum dynamics inside and classical optimization outside."

Exercises

1. For a simple 1-qubit ansatz $R_Y(\theta_2)R_X(\theta_1)|0\rangle$, identify the state, control input, dynamics, output, and objective in control-theoretic language.
2. The two-loop structure has different time scales (inner: microseconds, outer: seconds). Identify *which* control problems are most affected by this disparity. (Hint: think about adaptive control with measurement feedback.)
3. (VQA) For a quantum hardware system with a known calibration drift on the order of hours, sketch how a control-theoretic adaptive scheme would respond. Which loop (inner or outer) handles the drift?

Research Extensions

- **Quantum-classical hybrid control theory** — formalizing the two-loop structure with explicit theorems.
- **Real-time adaptive VQA controllers** — implementing feedback loops at the inner-loop time scale.
- **Robust VQA design via robust control techniques** — applying H_∞ and μ-synthesis to ansatz selection.
- **Receding-horizon (MPC) VQA optimization** — model-predictive control variants of VQA optimizers.

Cross-links to Other Courses

- **Classical Control Theory** — Wiener, Kalman, Pontryagin, the foundational literature.

- **Quantum Control (D'Alessandro)** — the existing field at the inner-loop level.
- **Dynamical Systems** — the broader framework of state-space dynamics.
- **Module 6 (Optimization Dynamics)** — the optimizer-theoretic view, complementary.
- **Module 10 (Adaptive Control Systems)** — the deployment endpoint.

Observability of Quantum States

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Control-Theoretic View **Node:** 7.2

In control theory, **observability** is the question: given the outputs you can measure, can you reconstruct the underlying state? It is the dual of **controllability** (which input combinations can drive the system to which states) — observability asks what *information* you have about a state, not what states you can reach.

For VQAs, observability translates into a specific operational question: **does the cost function uniquely identify the optimal parameters, or are there directions in parameter space along which the cost is flat (or symmetric)?** If the cost is observability-deficient, then no amount of optimization will distinguish between equivalent parameter values — there is a *gauge symmetry* and the optimizer wanders aimlessly along the gauge orbit.

This matters for VQA design in several distinct ways: it explains certain "stuck" optimizer behaviors, motivates the choice of cost function, and connects to the broader question of *why some VQAs are easier to train than others*.

This node introduces classical observability, the quantum-state observability problem, the cost-function observability question for VQAs, and how to diagnose observability deficiencies in practice.

Classical Observability

In classical control, a linear system is

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}, \quad \mathbf{y} = C\mathbf{x},$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state, $\mathbf{u} \in \mathbb{R}^m$ is the input, and $\mathbf{y} \in \mathbb{R}^p$ is the measured output.

Observability theorem (Kalman): the system is observable if and only if the **observability matrix**

$$\mathcal{O} = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

has rank n. Equivalently, no two distinct initial states $\mathbf{x}(0)$ and $\mathbf{x}'(0)$ produce identical output trajectories.

If the observability matrix has rank $r < n$, then there is an $(n-r)$-dimensional **unobservable subspace**: states that differ by a vector in this subspace are indistinguishable from output measurements.

This is the canonical setup. The **rank of the observability matrix** is the central quantitative measure.

Observability For Quantum States

The quantum analogue is more delicate. Quantum measurement has two distinct features:

1. Measurement collapses the state. Once you measure, the state changes. So the "output trajectory" of a single state is impossible — you can only measure one shot, then the state is gone.

2. Single-shot measurements are random. Even given a fixed observable, the result of a single measurement is sampled from a distribution.

So "observability" in the quantum sense means: **given the ability to prepare many copies of the same state and measure any of a fixed set of observables, can you reconstruct the state?**

The answer is: **yes, with sufficiently informative observables**. Specifically, a set of observables $\{O_1, O_2, \ldots, O_K\}$ is **informationally complete** if knowing all expectation values $\{\langle\psi|O_k|\psi\rangle\}_k$ uniquely identifies $|\psi\rangle$ (up to global phase).

Standard informationally complete sets:

- **Pauli decomposition:** all 4^n products of single-qubit Paulis $\{I, X, Y, Z\}^{\otimes n}$. Together they span all Hermitian operators on n qubits.
- **Symmetric informationally complete (SIC) POVMs:** d^2 outcomes that uniquely identify any d-dimensional state.
- **Mutually unbiased bases (MUBs):** $d + 1$ bases, each measurement gives d outcomes, together informationally complete.

Cost of informational completeness: 4^n Pauli observables (or $d^2 = 4^n$ SIC outcomes) — exponentially many. **Full quantum state reconstruction is exponentially expensive.**

In practice, you don't need full reconstruction. You only need enough observables to identify the *parameters of the ansatz*, which is a much smaller set (at most P numbers).

Cost-Function Observability For VQAs

The VQA-specific observability question:

Definition (cost observability). A cost function $C(\boldsymbol{\theta})$ is **observable** at $\boldsymbol{\theta}^*$ if no other parameter value gives the same cost in a neighborhood: $C(\boldsymbol{\theta}^* + \delta) \neq C(\boldsymbol{\theta}^*)$ for all $\delta \neq 0$ in some neighborhood.

More quantitatively: the cost is **first-order observable** if the gradient $\nabla C(\boldsymbol{\theta}^*)$ has full rank in directions that matter (i.e., is not zero along any direction you care about). It is **second-order observable** if the Hessian has full rank.

Observability deficiencies in VQAs:

1. **Gauge symmetries.** If the ansatz has parameter combinations that produce the same state (e.g., $R_Z(\theta_1)R_Z(\theta_2)$ only depends on $\theta_1 + \theta_2$), then the cost depends only on $\theta_1 + \theta_2$, and the perpendicular direction is unobservable. This is the *gauge orbit* problem.
2. **Symmetries of the cost.** If the cost is invariant under some parameter transformation (e.g., $\boldsymbol{\theta} \to -\boldsymbol{\theta}$ for some symmetric ansätze), then there are multiple equivalent minima and the cost can't distinguish them.
3. **Local degeneracies.** Even without explicit symmetries, the cost can have flat directions at specific points (e.g., saddles, ridges) where the gradient vanishes along some direction.
4. **Barren plateaus as an extreme case.** A barren plateau is a region where the gradient is *near zero in all directions* — the cost is unobservable in *every* direction simultaneously. This is the worst-case observability deficiency.

The Observability Matrix For VQAs

The classical observability matrix can be adapted to VQAs, with parameters playing the role of state.

For a cost function $C(\boldsymbol{\theta})$ viewed as a function on parameter space, the local observability is captured by the **Hessian matrix** $H_{ij}(\boldsymbol{\theta}) = \partial^2 C / \partial\theta_i \partial\theta_j$. The rank of H at a critical point measures the local observability:

- **Full rank P:** the critical point is non-degenerate. Locally, the cost uniquely identifies the parameters in all directions.
- **Rank $r < P$:** the critical point has $P - r$ flat directions. Locally, the cost is unobservable in those directions.
- **Rank 0:** the cost is locally flat — barren plateau.

The **observability gap** is the difference between the largest and smallest non-zero Hessian eigenvalues. A small gap means the optimizer struggles to distinguish important from unimportant directions, and the convergence rate degrades accordingly.

Observability And The Fubini-Study Metric

A more geometric view connects observability to the **Fubini-Study metric** on the projective Hilbert space.

For a parameterized state $|\psi(\boldsymbol{\theta})\rangle$, the FS metric on parameter space has matrix elements

$$g_{ij}(\boldsymbol{\theta}) = \mathrm{Re}\left[\langle \partial_i \psi | \partial_j \psi \rangle - \langle \partial_i \psi | \psi \rangle \langle \psi | \partial_j \psi \rangle\right].$$

This is the **Quantum Fisher Information Matrix** (QFIM, Module 6.2) and it captures how much the state changes per unit parameter change.

Connection to observability:

- If $g(\boldsymbol{\theta})$ is full rank, every parameter direction produces a distinguishable state — full state-level observability.
- If $g(\boldsymbol{\theta})$ is rank-deficient, there are parameter directions that don't change the state — *gauge directions*. These are unobservable in any cost function.
- The gauge dimension is $P - \mathrm{rank}(g)$.

This is a *state-level* observability that is independent of the cost. The cost-level observability (Hessian rank of C) is generally tighter (a flat direction in g is automatically flat in C, but a flat direction in C need not be flat in g).

The QFIM gives the **theoretical upper bound** on cost observability for any cost function built from $|\psi(\boldsymbol{\theta})\rangle$.

Practical Diagnostics

How do you *check* observability for a real VQA? A few standard diagnostics:

1. Hessian rank at a candidate optimum. Compute the Hessian (via second-order parameter shift), check the eigenvalue distribution. Rank $< P$ signals an observability deficiency.

2. QFIM rank. Compute the QFIM (Section 6.2), check rank. The gap to P gives the gauge-direction count.

3. Trajectory variance. Run the optimizer from multiple random starting points. If the final parameters cluster, the cost is observable. If they spread out (different parameters reaching the same cost), there are equivalent solutions.

4. Cost contour visualization. For low-dimensional cases, plot the cost contours. Flat regions = observability deficiency.

5. Gradient norm distribution. During training, track the gradient norm. If it becomes very small without the cost decreasing, you're in a flat region.

These diagnostics are simple in principle but require care in interpretation — many VQAs have *some* observability deficiency without it being catastrophic.

Fixing Observability Deficiencies

Once diagnosed, what can you do?

1. Remove gauge redundancies. If the ansatz has explicit gauge freedoms, parameterize the gauge slice instead. E.g., fix one of the redundant parameters to zero.

2. Choose a more informative cost. Some costs are observability-deficient by design (e.g., a single global expectation value). Adding multiple observables (Section 7.6 sketches the multi-observable case) can break the degeneracy.

3. Add a regularization term. A regularization term that depends on the parameters directly (rather than on the cost) can break flat directions and make the optimization landscape better-conditioned. This is one of the thesis's main themes.

4. Restart from a different initialization. Sometimes the observability deficiency is local — restarting from a different point lands you in a region with better observability.

5. Use an observability-aware optimizer. Natural gradient (Module 6.2) explicitly accounts for the QFIM and avoids moving in unobservable directions. This is a partial fix.

The Separation Of State And Parameter Observability

A subtle but important distinction:

State observability: can the cost function identify the state $|\psi(\boldsymbol{\theta})\rangle$ — i.e., does the cost uniquely depend on the state?

Parameter observability: can the cost function identify the parameters $\boldsymbol{\theta}$ — i.e., does the cost uniquely depend on the parameters?

These are *not* the same. A cost can be state-observable (different states give different costs) but parameter-deficient (different parameters give the same state, hence the same cost). This is exactly the gauge-orbit case.

Conversely, a cost can be parameter-observable but state-deficient (different parameters give different costs even though they produce different but cost-equivalent states). This is rare but possible with badly designed costs.

For practical VQA design, **state observability is what you want at the application level** (the optimizer should find the right state), but **parameter observability is what the optimizer sees** (it can only adjust parameters, not states directly). The mismatch between them is a recurring source of subtle bugs and slow convergence.

Structural Role

This node is the **measurement-side foundation** of Module 7. It introduces the observability concept, walks through the classical and quantum versions, applies it to the VQA cost-observability question, and provides diagnostics for observability deficiencies.

It directly enables node 7.4 (optimal control theory, which uses observability to design state estimators) and node 7.6 (feedback control, which requires observable outputs to close the loop).

Relations to Other Concepts

- **Kalman (1960)** — the original classical observability theorem.
- **D'Alessandro (2007)** — quantum observability via informationally complete observables.
- **Quantum tomography** — the operational version of state observability.
- **Module 6.2 (Natural Gradient)** — the QFIM, which captures state observability.
- **Module 8 (Expressivity)** — the dual question (controllability rather than observability).

Worked Example — Observability Of A 1-Qubit Ansatz

Consider $|\psi(\theta_1, \theta_2)\rangle = R_Y(\theta_1)R_Y(\theta_2)|0\rangle$ — two R_Y rotations on 1 qubit.

State. $R_Y(\theta_1)R_Y(\theta_2) = R_Y(\theta_1 + \theta_2)$. So the state depends only on $\theta_1 + \theta_2$.

Cost. Suppose $C = \langle\psi|Z|\psi\rangle = \cos(\theta_1 + \theta_2)$. The cost depends only on the sum.

Observability analysis. - **State observability:** the cost uniquely identifies the state (since cost = cos of the rotation angle, which is monotone in a half-period). - **Parameter observability:** the cost is *constant* along the line $\theta_1 + \theta_2 = \text{const}$. This is a 1-dimensional unobservable direction in 2-dimensional parameter space. - **Gauge dimension:** 1 (the $\theta_1 - \theta_2$ direction is unobservable).

QFIM. Let $\phi = \theta_1 + \theta_2$. Then $|\psi\rangle = R_Y(\phi)|0\rangle$ and $|\partial_1\psi\rangle = |\partial_2\psi\rangle = (\partial/\partial\phi)|\psi\rangle$. The QFIM is

$$g = \begin{pmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{pmatrix},$$

which has rank 1, eigenvalues $\{1/2, 0\}$. The zero eigenvalue confirms the unobservable direction.

Hessian. $H_{ij} = \partial^2 C/\partial\theta_i\partial\theta_j = -\cos(\theta_1 + \theta_2)$ for all i, j. Same matrix structure (rank 1).

Practical implication. A gradient-descent optimizer started at random will move in the gradient direction (which is along $(1, 1)^T$), reaching a cost-decreasing trajectory but never resolving the perpendicular direction. The final (θ_1, θ_2) will not be unique — it will lie on the line $\theta_1 + \theta_2 = \theta^*$ for some θ^*.

Fix. Eliminate θ_2 (set to zero) and use only θ_1. Now you have a 1-parameter ansatz with full observability. Same expressive power, half the parameters, no gauge problem.

This worked example is small but the structural lesson generalizes: **redundant parameters create observability holes**, and pruning them makes the optimization easier.

Visualization

Pending. A 2D heatmap of $C(\theta_1, \theta_2) = \cos(\theta_1 + \theta_2)$ showing the diagonal stripes of constant cost. Overlaid: a gradient-descent trajectory that follows the gradient direction but ends at an arbitrary point on a stripe. Caption: "The flat direction is the unobservable subspace. The optimizer doesn't know where on the stripe to stop."

Exercises

1. For a 2-qubit ansatz $R_Y(\theta_1) \otimes R_Y(\theta_2)|00\rangle$ with cost $C = \langle Z_0 + Z_1\rangle$, is the cost observable in both parameters?
2. The QFIM has rank r at a parameter point. How many gauge directions does the parameter space have at that point?

3. (VQA) For a Hamiltonian Variational Ansatz on 6 qubits with 12 parameters, suppose you compute the Hessian rank at a candidate optimum and find it is 9. What does this imply about the optimizer behavior?

Research Extensions

- **Dynamical observability for time-varying systems** — extending classical observability theorems to drift-prone hardware.
- **Approximate observability under shot noise** — how many shots are needed to identify a parameter to a given accuracy?
- **Cost-design for observability** — algorithmic methods to choose costs that maximize parameter observability.
- **Observability and adaptive optimization** — using observability diagnostics to drive optimizer choice.

Cross-links to Other Courses

- **Linear Systems Theory** — Kalman observability, the classical foundation.
- **Quantum State Tomography** — the operational version of state-level observability.
- **Module 6.2 (Natural Gradient)** — the QFIM-based view of observability.
- **Module 7.3 (Controllability)** — the dual notion.
- **Module 7.6 (Feedback Control)** — where observability becomes operationally critical.

Controllability of Parameterized Circuits

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Control-Theoretic View **Node:** 7.3

This is the **dual** of node 7.2. Where observability asks "which states can we *learn about*?", controllability asks "which states can we *steer to*?"

For a VQA, controllability is the question of which states are *reachable* by varying the circuit parameters. This is the same question Module 8 asked under the name "expressivity," but the control-theoretic view gives it a different mathematical form (Lie algebras of vector fields, accessibility conditions, Chow-Rashevsky theorem) and a more *dynamical* flavor.

The two perspectives — expressivity (Module 8) and controllability (this node) — are equivalent in their conclusions but they are formulated differently and they each give different intuitions and proof techniques. This node walks through the controllability formulation, makes the link to expressivity explicit, and applies the controllability theorems to the practical question of ansatz design.

Classical Controllability

For a linear classical system

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u},$$

the **controllability matrix** is

$$\mathcal{C} = \begin{pmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{pmatrix}.$$

Kalman controllability theorem: the system is controllable (i.e., from any initial state $\mathbf{x}_0$ you can reach any target state $\mathbf{x}_f$ in finite time using suitable $\mathbf{u}(t)$) if and only if $\mathcal{C}$ has full rank n.

If the rank is $r < n$, then there is an $(n-r)$-dimensional **uncontrollable subspace**: states that differ by a vector in this subspace cannot be reached from each other.

This is the linear version. For nonlinear systems, the controllability question is more delicate and uses the language of Lie algebras of vector fields.

Nonlinear Controllability And The Lie Bracket

For a nonlinear control-affine system

$$\dot{\mathbf{x}} = f_0(\mathbf{x}) + \sum_{j=1}^{m} u_j f_j(\mathbf{x}),$$

where f_0 is the **drift vector field** (autonomous dynamics with no control) and $f_1, \ldots, f_m$ are the **control vector fields**, the relevant object is the **Lie algebra generated by the vector fields**.

Lie bracket of two vector fields:

$$[f_i, f_j](\mathbf{x}) = \frac{\partial f_j}{\partial \mathbf{x}}(\mathbf{x}) f_i(\mathbf{x}) - \frac{\partial f_i}{\partial \mathbf{x}}(\mathbf{x}) f_j(\mathbf{x}).$$

The Lie bracket measures the "non-commutation" of two vector fields. Operationally, if you flow along f_i for a small time, then f_j for a small time, then $-f_i$, then $-f_j$, you don't return to the start — you arrive at a point displaced from the start by approximately $[f_i, f_j]$ times the square of the time step.

Chow-Rashevsky theorem: the system is controllable on a connected manifold if the Lie algebra generated by $\{f_0, f_1, \ldots, f_m\}$ under repeated commutation spans the tangent space at every point.

This is the generalization of Kalman's controllability theorem to nonlinear systems. The "rank" condition is replaced by a Lie-algebra spanning condition.

Quantum Controllability

The Schrödinger equation with control Hamiltonian $H(\mathbf{u}(t)) = H_0 + \sum_j u_j(t) H_j$ is

$$i|\dot{\psi}\rangle = \left(H_0 + \sum_j u_j(t) H_j \right) |\psi\rangle.$$

This is a control-affine system on the projective Hilbert space, with drift H_0 and control vector fields generated by H_j.

Theorem (Schirmer-Solomon, D'Alessandro): the system is **fully controllable on the unitary group** $U(2^n)$ (i.e., any unitary can be implemented with suitable $\mathbf{u}(t)$) if and only if the Lie algebra generated by $\{iH_0, iH_1, \ldots, iH_m\}$ under commutation is the full algebra $\mathfrak{u}(2^n)$.

This is the dynamical Lie algebra (DLA) introduced in node 8.1. The controllability theorem and the expressivity theorem are *the same theorem* in different language.

Variants:

- **State controllability (rather than unitary controllability):** the system is state-controllable if any state can be reached from any other. Weaker than full unitary controllability — requires the DLA to act *transitively* on the state space, not generate the full unitary group.
- **Operator controllability:** any unitary in some subgroup (e.g., particle-number-conserving) can be implemented. Requires the DLA to be exactly that subgroup's algebra.
- **Density-matrix controllability:** any density matrix (mixed states) can be reached. Requires *more* than full unitary controllability — you also need access to non-unitary operations (measurement, dissipation).

Controllability And Expressivity Are The Same Thing

The link is exact. For a parameterized circuit $U(\boldsymbol{\theta}) = \prod_j e^{-i\theta_j H_j}$:

- **Expressivity** = dimension of the reachable manifold = dimension of the orbit of $|0\rangle$ under the group generated by $\{e^{-i\theta H_j}\}$.
- **Controllability** = dimension of the orbit reachable under the control system with control Hamiltonians $\{H_j\}$.

These are the same dimension. The DLA $\mathfrak{g} = \mathrm{Lie}(\{iH_j\})$ governs both:

- $\dim \mathcal{R}(U) = \dim(\mathfrak{g} \cdot |0\rangle)$ (the orbit of $|0\rangle$).
- $\dim(\text{controllable set}) = \dim(\mathfrak{g} \cdot |0\rangle)$ (same).

So **the controllability theorem of quantum control theory and the expressivity theorem of VQA theory are the same theorem in different language**, both reducing to the dimension of the DLA.

This is more than a notational point — it means **70 years of quantum control theory directly applies to VQA expressivity questions**, including:

- Algorithmic methods for computing the DLA (graph-theoretic Lie closure).
- Conditions for *small* DLAs (when symmetries reduce the algebra).
- Approximate controllability theorems (when you can't reach the whole orbit but get arbitrarily close).
- Time-optimal control (the minimum time to reach a target — analogous to minimum circuit depth).

The thesis's view: **VQA practitioners should be reading the quantum control literature**, because that's where the relevant theorems live.

Reachable Set Vs Reachability In Finite Time

A subtle but important distinction:

Reachable set: the set of states you can *eventually* reach, possibly with arbitrarily long control trajectories.

Time-bounded reachable set: the set of states you can reach within a fixed time bound T.

For a fully controllable system, the reachable set is the whole projective Hilbert space. But the *time-bounded* reachable set is much smaller — it grows with T and only equals the full space in the limit $T \to \infty$.

For VQAs, "time" is replaced by **circuit depth**. The depth-bounded reachable set is the set of states reachable by a circuit of at most L layers. As L increases, this grows monotonically toward the full reachable set.

Quantitative version: for an HEA with random initialization, the depth-bounded reachable set covers an ϵ-fraction of the Hilbert space at depth $L \sim n \log(1/\epsilon)$. Below this depth, the reachable set is a sparse subset; above it, the ansatz is approximately fully expressive.

This is the bridge between the *algebraic* expressivity (DLA dimension, asymptotic) and the *practical* expressivity (what depth do I actually need?).

Approximate And Conditional Controllability

For real ansätze, exact controllability is often unattainable, but approximate or conditional versions hold:

1. Approximate controllability. The reachable set is dense in the target manifold but not equal to it. Practically, you can get *arbitrarily close* to any target state but not exactly reach it.

2. Conditional controllability. The reachable set is the full target manifold *given* certain initial conditions (e.g., starting from a specific state, or with parameters in a specific range).

3. Bounded-depth controllability. The reachable set at depth L is some subset of the full target manifold, growing with L.

4. Symmetry-restricted controllability. The reachable set is the orbit of a symmetry group. If your problem has the symmetry, this is enough.

For VQAs, **bounded-depth controllability with symmetry restriction** is the typical practical regime. You don't need to be able to reach every state in Hilbert space — you need to be able to reach every state in the symmetry-allowed sector that contains your problem's solution.

Practical Diagnostics For Controllability

How do you check if your ansatz is controllable for the problem at hand?

1. Compute the DLA. Symbolically close $\{iH_j\}$ under commutators and count the dimension. If the DLA dimension matches the expected target manifold dimension, you have full controllability.

2. Check symmetry conservation. Identify the conserved quantities (operators that commute with all H_j). The reachable set is restricted to the joint eigenspaces of these conserved quantities. Make sure your target state is in the correct eigenspace.

3. Empirical reachability test. Sample many target states from the desired distribution and check whether your ansatz can approximate each one (via local optimization). If most can be approximated, you're effectively controllable.

4. Random parameter coverage. Sample many random parameter values and check the *variety* of states produced. Low variety = limited reachable set.

These diagnostics are operational versions of the formal Lie-algebraic conditions, suitable for ansätze where exact computation of the DLA is not feasible.

Controllability And Trainability — Revisited

A key insight from Module 8 (and the Ragone et al. theorem):

Trainability is inversely proportional to controllability.

- Fully controllable ansatz (DLA = $\mathfrak{u}(2^n)$): unique answer reachable, but barren plateaus everywhere.
- Restricted-controllability ansatz (DLA = small Lie subalgebra): reachable set is smaller, but trainability is better.

The control-theoretic framing makes this very clean: **controllability and trainability are conflicting design objectives, with the DLA dimension as the single parameter that controls both**.

This is the same tradeoff that Module 8.5 stated in expressivity language. The control language gives it a slightly different flavor:

- **Expressivity language:** "How big is the reachable manifold? Bigger = harder to optimize."
- **Control language:** "Which states can the control system reach? More reachable = noisier feedback signal."

Both are correct. The choice is rhetorical.

Time-Optimal Control And Minimum Circuit Depth

A particularly nice application of control theory to VQAs: **time-optimal quantum control**.

Problem. Given a target unitary U_T and a set of control Hamiltonians $\{H_j\}$, find the **minimum time** T^* to implement U_T (within tolerance) using piecewise-constant controls.

Theorem (Khaneja-Brockett-Glaser 2001). For two-qubit systems with a specific control Hamiltonian, the minimum time has an explicit closed form in terms of the singular values of U_T. This is one of the few cases where time-optimal control of quantum systems is solved exactly.

Implication for VQAs. The minimum circuit depth to implement a target state is the discrete-time analogue. Time-optimal control gives **lower bounds** on circuit depth: no ansatz can implement the target faster than the time-optimal control trajectory.

For VQA design, this is useful: it gives you a *target depth* below which you should not bother trying. If your ansatz is at this depth or above, the question is whether the *parameter optimization* can find the right control trajectory.

Structural Role

This node is the **reachability foundation** of Module 7. It introduces classical and quantum controllability, identifies the DLA-based conditions, and establishes the equivalence between control-theoretic controllability and VQA expressivity.

It is the dual of node 7.2 (observability) and a prerequisite for node 7.4 (optimal control), which uses controllability to formulate the cost-minimization problem.

Relations to Other Concepts

- **Chow-Rashevsky theorem** — nonlinear controllability via Lie algebras.
- **Schirmer-Solomon (2001)** — quantum controllability theorem.
- **D'Alessandro (2007)** — *Introduction to Quantum Control and Dynamics.*
- **Khaneja-Brockett-Glaser (2001)** — time-optimal two-qubit control.
- **Module 8.1 (Expressivity)** — the same theorem in different language.

Worked Example — Controllability Of A 2-Qubit XX Ansatz

Consider $U(\theta_1, \theta_2, \theta_3) = e^{-i\theta_3 X_0 X_1} R_Y(\theta_2)_1 R_Y(\theta_1)_0$ — single-qubit R_Y on each qubit followed by an XX coupling.

Generators: - $H_1 = Y_0$ (generator of R_Y on qubit 0). - $H_2 = Y_1$ (generator of R_Y on qubit 1). - $H_3 = X_0 X_1$ (generator of the XX coupling).

Compute the DLA. Take Lie brackets: - $[H_1, H_3] = [Y_0, X_0 X_1] = -2iZ_0 X_1$ (new operator). - $[H_2, H_3] = [Y_1, X_0 X_1] = -2iX_0 Z_1$ (new operator). - $[H_1, [H_1, H_3]] = ...$ more new operators.

Continuing, the DLA closes after several steps and contains: $\{Y_0, Y_1, X_0 X_1, Z_0 X_1, X_0 Z_1, Z_0 Y_1, Y_0 Z_1, Z_0 Z_1, X_0, X_1, ...\}$.

For 2 qubits, the full Lie algebra $\mathfrak{u}(4)$ has dimension 16. The DLA of this ansatz, after closure, is exactly all of $\mathfrak{u}(4)$ — so it is **fully controllable**.

Implication. This 3-parameter ansatz (after enough repetitions) can reach any 2-qubit state. State-universal at sufficient depth.

Trainability. Because the DLA is full $\mathfrak{u}(4)$, the Ragone bound predicts gradient variance $\sim 1/16$ — small but not exponentially small. For 2 qubits, this is fine. For n qubits with similar structure, the bound becomes $\sim 1/4^n$ — barren plateau.

This is a clean instance where controllability theory gives both a positive result (universality) and a negative result (untrainability at scale) from the same calculation.

Visualization

Pending. A diagram showing two ansätze: (a) a controllable one (DLA = full algebra) with reachable set = full Hilbert space; (b) an uncontrollable one with reachable set = a low-dimensional submanifold. Caption: "Controllability is about which states you can steer to."

Exercises

1. For a 1-qubit system with control Hamiltonians $\{X, Z\}$, is it fully controllable? Compute the DLA.
2. For a 2-qubit system with control Hamiltonians $\{X_0, X_1, Z_0Z_1\}$, is it fully controllable? (Hint: this is the standard QAOA-style set.)
3. (VQA) For a 4-qubit system with only $\{Y_0, Y_1, Y_2, Y_3, Z_0Z_1, Z_1Z_2, Z_2Z_3\}$ (linear chain XX-equivalent), compute the DLA dimension. Is the ansatz fully controllable?

Research Extensions

- **Efficient DLA computation** — algorithmic methods for large ansätze.
- **Approximate controllability with explicit error bounds** — quantitative versions of "almost controllable."
- **Time-optimal VQA depth** — combining time-optimal quantum control with VQA design.
- **Controllability under noise** — how does decoherence affect the reachable set?

Cross-links to Other Courses

- **Lie Theory** — Lie algebras, Lie groups, Lie brackets.
- **Differential Geometry** — vector fields, integral curves, manifolds.
- **Classical Control Theory** — Chow-Rashevsky, Kalman controllability.
- **Module 8.1 (Expressivity Theory)** — the same content in expressivity language.
- **Module 7.4 (Optimal Control)** — uses controllability as a precondition.

Optimal Control Theory

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Control-Theoretic View **Node:** 7.4

This is the **central technical** node of Module 7. Where 7.2 and 7.3 introduced observability and controllability as *structural* properties, optimal control theory is the framework for *choosing* the control input that best achieves a goal. It is the variational calculus of control systems, and it is what justifies casting VQA optimization as a control problem in the first place.

The connection is direct: **VQA parameter optimization is a discrete-time optimal control problem**. Every theorem and algorithm in optimal control theory has a VQA analogue. This node walks through the framework, the major algorithmic families (LQR, indirect methods, direct methods, GRAPE), and the specific quantum control variants (Krotov, CRAB, gradient ascent pulse engineering).

The thesis's view is that **the gap between optimal control theory and VQA optimization is mostly notational** — the techniques that have been developed for half a century in classical and quantum control are exactly the techniques VQA practitioners are slowly rediscovering. Module 7 makes the import explicit.

The Optimal Control Problem

The general setup:

- **State:** $\mathbf{x}(t) \in \mathcal{X}$, with dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$.
- **Control:** $\mathbf{u}(t) \in \mathcal{U}$, to be chosen.
- **Initial condition:** $\mathbf{x}(0) = \mathbf{x}_0$.
- **Terminal time:** T (fixed or free).
- **Cost functional:**

$$J[\mathbf{u}] = \phi(\mathbf{x}(T)) + \int_0^T L(\mathbf{x}(t), \mathbf{u}(t), t)\, dt,$$

where ϕ is a **terminal cost** and L is a **running cost**.

Problem. Find $\mathbf{u}^*(t)$ that minimizes J, subject to the dynamics and any constraints on $\mathbf{x}, \mathbf{u}$.

This is the **canonical optimal control problem**. It generalizes the calculus of variations (where there is no control input separate from the state) and is the parent framework for everything in this node.

Linear Quadratic Regulator (LQR)

The simplest non-trivial case: linear dynamics, quadratic cost, no constraints.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}, \quad J = \mathbf{x}(T)^T S \mathbf{x}(T) + \int_0^T (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})\, dt.$$

Theorem. The optimal control is a linear feedback law $\mathbf{u}^*(t) = -K(t)\mathbf{x}(t)$, where $K(t) = R^{-1}B^T P(t)$ and $P(t)$ satisfies the **Riccati differential equation**

$$-\dot{P} = A^T P + PA - PBR^{-1}B^T P + Q, \quad P(T) = S.$$

Implications:

1. **The optimal control is linear in the state** — a clean closed-form result.
2. **The Riccati equation is solvable in closed form** for many cases — gives explicit control laws.
3. **The result generalizes** to discrete time, infinite horizon, and stochastic noise (Linear Quadratic Gaussian, LQG).

For VQAs: the analogue is "linear ansatz, quadratic cost." Both assumptions are wrong for typical VQAs (the ansatz is highly nonlinear in parameters; the cost is non-convex). So LQR doesn't apply directly. But it serves as the **theoretical limit case** — the easiest possible optimal control problem, against which to measure the difficulty of harder ones.

Indirect Methods: The Hamiltonian Formulation

For nonlinear problems, the standard approach is via the **Pontryagin minimum principle** (covered in detail in node 7.5). The idea: introduce a **costate** $\boldsymbol{\lambda}(t)$ (Lagrange multiplier for the dynamics constraint) and form the control Hamiltonian

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t) = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T f(\mathbf{x}, \mathbf{u}, t).$$

The optimal trajectory satisfies:

$$\dot{\mathbf{x}} = \partial H/\partial \boldsymbol{\lambda}, \quad \dot{\boldsymbol{\lambda}} = -\partial H/\partial \mathbf{x}, \quad \mathbf{u}^* = \arg\min_{\mathbf{u}} H.$$

This is a **two-point boundary value problem** (the state is given at $t = 0$, the costate at $t = T$) — harder to solve numerically than an initial-value problem, but in principle gives the exact optimum.

Algorithms: shooting methods, multiple shooting, collocation. All require careful initialization because the Hamiltonian formulation is sensitive to numerical issues.

For VQAs, the indirect approach corresponds to writing down the parameter-shift gradient conditions and solving them as a system of nonlinear equations. Practically, this is rarely done — gradient descent (a direct method) is simpler.

Direct Methods: Discretize First, Optimize Second

The alternative: discretize the problem in time and turn it into a finite-dimensional optimization.

Procedure: 1. Discretize time into N intervals: $t_k = k\Delta t$. 2. Approximate the state and control as piecewise functions: $\mathbf{x}_k = \mathbf{x}(t_k), \mathbf{u}_k = \mathbf{u}(t_k)$. 3. Discretize the dynamics: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot f(\mathbf{x}_k, \mathbf{u}_k)$ (Euler) or higher-order schemes. 4. Discretize the cost: $J \approx \phi(\mathbf{x}_N) + \Delta t \sum_k L(\mathbf{x}_k, \mathbf{u}_k)$. 5. Optimize over $\{\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}\}$ directly, treating the dynamics as constraints (or implicit functions).

This is a **constrained nonlinear program** with Nm variables (where m is the control dimension). Solvable by gradient methods, sequential quadratic programming, or interior point.

For VQAs: this is exactly what VQA optimization is. The control intervals are the gates; the control variables are the parameters; the dynamics are the unitary evolution; the cost is the final expectation value. The "discretize first, optimize second" approach is the unstated framework of all VQA papers.

Quantum Optimal Control: GRAPE

The dominant algorithm for *continuous-pulse* quantum optimal control: **Gradient Ascent Pulse Engineering** (Khaneja et al. 2005).

Setup. Time evolution under a Hamiltonian $H(t) = H_0 + \sum_j u_j(t) H_j$. Discretize $u_j(t)$ into N piecewise-constant intervals. The total unitary is

$$U_{\text{total}} = \prod_{k=0}^{N-1} \exp\left(-i\Delta t \left(H_0 + \sum_j u_{j,k} H_j\right)\right).$$

Goal. Find $\{u_{j,k}\}$ that maximizes the fidelity $|\text{Tr}(U_T^\dagger U_{\text{total}})|^2/d^2$ to a target unitary U_T, or that prepares a target state.

GRAPE update rule. Compute the gradient of the fidelity with respect to each $u_{j,k}$ using the analytical formula

$$\frac{\partial F}{\partial u_{j,k}} = -i\Delta t \cdot \text{Re}\left[\text{Tr}\left(\rho^\dagger H_j U_{\text{total}}\right)\right] + O(\Delta t^2),$$

where ρ is a reference state, then take a gradient step.

Compared to VQA parameter-shift: GRAPE uses an *approximate* gradient (the $O(\Delta t^2)$ error term) but is much faster per evaluation (no parameter-shift, just one forward simulation). For continuous quantum control, this tradeoff is worth it. For VQAs (where parameter-shift is exact and shot noise is the bottleneck), the tradeoff is different.

The link: GRAPE on a fine discretization is equivalent to VQA optimization with an HEA-like ansatz. The same algorithm, just at a different level of abstraction.

Krotov And CRAB

Two other major quantum control algorithms.

Krotov

A **monotonic** algorithm: each iteration is guaranteed to decrease the cost (no step size to tune). Works by iteratively solving a coupled system of forward (state) and backward (costate) equations.

Strengths: monotonic convergence, no step size, theoretically clean. **Weaknesses:** slower per iteration than GRAPE, harder to implement, requires the full Hamiltonian to be known.

CRAB (Chopped Random Basis)

Parameterizes the control $u_j(t)$ in a *low-dimensional* basis (e.g., truncated Fourier series) and optimizes over the basis coefficients.

Strengths: few parameters, robust to local minima, easy to implement. **Weaknesses:** limited expressivity (can only represent functions in the chosen basis).

Connection to VQAs: CRAB is essentially "VQA with a smart parameterization." The choice of basis is the analogue of ansatz design. CRAB has been successful in experimental quantum control settings, and its lessons (smart parameterization, low-dimensional optimization) translate directly to ansatz design for VQAs.

The VQA Optimal Control View

Putting it all together, here is the **VQA optimal control view**:

VQA concept	Optimal control concept
Parameters $\boldsymbol{\theta}$	Discretized control input $\{u_{j,k}\}$
Ansatz $U(\boldsymbol{\theta})$	Discretized propagator $U_{\text{total}}(\mathbf{u})$
Cost $C(\boldsymbol{\theta})$	Terminal cost $\phi(\mathbf{x}(T))$
Parameter-shift gradient	Exact GRAPE-like gradient
Optimizer (Adam, etc.)	Outer loop of GRAPE/Krotov
Ansatz design	Choice of control parameterization (CRAB basis)
Barren plateau	Vanishing gradient in the control optimization
Symmetry restriction	Constrained optimization in the control space

Implication: every VQA optimization technique has a quantum optimal control analogue, and vice versa. The fields developed in parallel without much cross-talk, and the bridge is exactly this node.

The thesis emphasizes the bridge because it gives access to:

1. **Convergence theorems** from optimal control (e.g., Krotov's monotonicity).
2. **Robustness techniques** from robust optimal control (H_∞, μ-synthesis).
3. **Time-optimal results** from optimal control (Khaneja-Brockett-Glaser for two-qubit gates).
4. **Adaptive control schemes** from adaptive optimal control (Module 10).

Specific Theorems Worth Knowing

A few theorems from optimal control with direct VQA application:

1. Pontryagin minimum principle (covered in 7.5). Necessary conditions for optimality. Applies to any optimal control problem with smooth dynamics and cost.

2. Hamilton-Jacobi-Bellman equation. A PDE for the value function (the optimal cost-to-go). Sufficient conditions for optimality. Solving it directly is usually intractable, but it justifies dynamic programming approaches.

3. Bellman's principle of optimality. "An optimal policy has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." The basis of dynamic programming.

4. Lyapunov stability. Conditions for the control system to converge to a desired state. For VQAs, gives criteria for when an optimizer is guaranteed to converge.

5. Separation principle. For LQG problems, the optimal controller can be decomposed into a state estimator and a feedback law. Not directly applicable to nonlinear quantum problems but inspires the architecture of adaptive VQAs.

These theorems are all in standard optimal control textbooks (e.g., Lewis-Vrabie-Syrmos, Bertsekas). Most VQA practitioners don't read them, which is a missed opportunity.

Limitations Of The Optimal Control Framing

The optimal control view has limitations:

1. Stochasticity. Standard optimal control assumes deterministic dynamics. VQAs have stochastic measurements, which makes them stochastic optimal control problems — a more complex framework.

2. High dimensionality. Optimal control techniques are designed for low-dimensional state spaces ($\mathcal{X} = \mathbb{R}^n$ for small n). The quantum state space has dimension 2^n, exponentially large.

3. Discrete nature. VQAs have discrete circuit structure (gates), not continuous controls. Some optimal control results don't translate cleanly.

4. Non-convexity. Optimal control results that assume convexity (LQR, certain Riccati decompositions) don't apply.

5. Hardware-specific constraints. Real VQA controls are constrained by gate fidelities, connectivity, and timing in ways that classical optimal control doesn't model.

These limitations mean the import is *partial* — not every theorem applies. But the framework, vocabulary, and most of the algorithmic ideas do apply, and the exceptions are usually known and discussed.

Structural Role

This node is the **algorithmic core** of Module 7. It introduces optimal control theory in its full generality, surveys the major algorithmic families (LQR, indirect, direct, GRAPE, Krotov, CRAB), and establishes the explicit mapping from optimal control concepts to VQA concepts.

It is the prerequisite for node 7.5 (Pontryagin minimum principle, the necessary conditions) and 7.6 (feedback control, the closed-loop generalization).

Relations to Other Concepts

- **Pontryagin et al. (1962)** — *The Mathematical Theory of Optimal Processes*, the foundational text.
- **Khaneja et al. (2005)** — GRAPE algorithm.
- **Krotov (1995)** — monotonic optimal control method.
- **Doria et al. (2011)** — CRAB algorithm.
- **Bellman (1957)** — *Dynamic Programming*, the principle of optimality.

Worked Example — VQA As GRAPE

Consider a 4-qubit VQA with HEA ansatz, depth 5, total parameters $P = 60$. Cost: ground-state energy of an H_2 Hamiltonian.

As an optimal control problem:

- **State:** $|\psi(t)\rangle \in \mathbb{C}^{16}$, evolving under the discretized circuit.
- **Control variables:** the 60 parameters, partitioned into time intervals (one parameter per gate).
- **Dynamics:** unitary evolution under the gate Hamiltonians.
- **Terminal cost:** $\phi = \langle\psi(T)|H_{H_2}|\psi(T)\rangle$.
- **Running cost:** zero (no path penalty).

As a GRAPE problem: with continuous-time control, this would be solved by computing the gradient of the fidelity at each time interval and gradient-stepping. The discretized version is exactly the parameter-shift VQA optimizer.

Differences from standard GRAPE:

1. The "time" is discrete (gates) rather than continuous (pulses).
2. The gradient is exact (parameter shift) rather than approximate (GRAPE finite difference).
3. The cost is estimated from finite shots, not computed exactly.

Differences captured by the same framework: all the GRAPE convergence theory, monotonicity guarantees, basin-hopping initialization, and warm-start strategies apply with minor modifications.

This is the kind of analysis the optimal control framing makes natural and the optimization-only framing misses.

Visualization

Pending. A flow diagram showing the optimal control problem (state, control, dynamics, cost, optimizer loop) with VQA labels in parentheses. Caption: "The same diagram, two vocabularies."

Exercises

1. For an LQR problem with $A = 0, B = 1, Q = 1, R = 1, S = 0$, solve the Riccati equation and write down the optimal feedback law.
2. The GRAPE gradient formula has an $O(\Delta t^2)$ error. Why is this acceptable for continuous control but not for VQA parameter-shift?
3. (VQA) For a QAOA-style ansatz with cost and mixer Hamiltonians, write the optimal control formulation explicitly. What is the running cost?

Research Extensions

- **VQA optimization via Krotov** — applying monotonic methods to VQA loss functions.
- **CRAB-style basis design for VQA** — choosing parameterization bases that improve trainability.
- **Stochastic optimal control for VQAs** — formal extensions to handle shot noise.
- **HJB equation for VQA value function** — partial dynamic-programming approaches.

Cross-links to Other Courses

- **Calculus of Variations** — the mathematical heritage of optimal control.
- **Pontryagin Minimum Principle** — the necessary conditions, in node 7.5.
- **Quantum Optimal Control** — GRAPE, Krotov, CRAB.
- **Module 6 (Optimization Dynamics)** — the optimization-only view.
- **Module 7.5 (Pontryagin)** — the next node.

Pontryagin Minimum Principle

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Control-Theoretic View **Node:** 7.5

The **Pontryagin minimum principle** (PMP) is the central theorem of classical optimal control theory. It gives **necessary conditions for a control trajectory to be optimal** — a generalization of the calculus-of-variations Euler-Lagrange equations to problems with control inputs.

For VQAs, the PMP has both theoretical and practical importance:

1. **Theoretically**, it tells you what an optimum *must* look like — every optimal VQA parameter setting satisfies a specific structural condition derived from the PMP.
2. **Practically**, it gives a *different* way to derive gradient conditions and update rules. The PMP-derived gradient is equivalent to the parameter-shift gradient, but the derivation is cleaner and connects to the broader optimal control literature.
3. **Conceptually**, the PMP makes the "two-point boundary value" structure of VQA optimization explicit, which clarifies why some VQAs are easy to optimize and others are not.

This node walks through the PMP, applies it to VQAs, and discusses the bang-bang structure it predicts for QAOA-like ansätze.

The Statement Of The Pontryagin Minimum Principle

Setup. Consider the optimal control problem

$$\min_{\mathbf{u}} J = \phi(\mathbf{x}(T)) + \int_0^T L(\mathbf{x}, \mathbf{u}, t)\, dt, \quad \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t), \quad \mathbf{x}(0) = \mathbf{x}_0.$$

Define the control Hamiltonian:

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t) = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T f(\mathbf{x}, \mathbf{u}, t).$$

Here $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$ is the **costate** (or "adjoint") variable — a Lagrange multiplier for the dynamics constraint.

Theorem (Pontryagin et al. 1962). If $\mathbf{u}^*(t)$ is optimal and $\mathbf{x}^*(t)$ is the corresponding state trajectory, there exists a costate $\boldsymbol{\lambda}^*(t)$ such that:

1. **State equation:** $\dot{\mathbf{x}}^* = \partial H / \partial \boldsymbol{\lambda}$.
2. **Costate equation:** $\dot{\boldsymbol{\lambda}}^* = -\partial H / \partial \mathbf{x}$.
3. **Minimum condition:** $\mathbf{u}^*(t) = \arg\min_{\mathbf{u}} H(\mathbf{x}^*(t), \mathbf{u}, \boldsymbol{\lambda}^*(t), t)$.
4. **Boundary conditions:** $\mathbf{x}^*(0) = \mathbf{x}_0$ (initial), $\boldsymbol{\lambda}^*(T) = \partial \phi / \partial \mathbf{x}|_{\mathbf{x}(T)}$ (terminal).

Interpretation: the PMP says optimal trajectories are characterized by a *two-point boundary value problem* in the state and costate, and at every instant the control minimizes the Hamiltonian. The costate plays the role of "marginal value of an extra unit of state" — it's the sensitivity of the cost-to-go with respect to the current state.

Why The PMP Is Useful

A few reasons:

1. It is necessary for optimality. Any optimal trajectory satisfies the PMP. So if you find a trajectory that does *not*, it's not optimal.

2. It reduces an infinite-dimensional problem to ODEs. The original problem is over function spaces (the control $\mathbf{u}(t)$), but the PMP gives a finite-dimensional system (state, costate, and a pointwise minimization).

3. It gives structural information. Even without solving, you can read off properties: when the Hamiltonian is linear in the control, the optimum is **bang-bang** (control jumps between extreme values); when it's quadratic, the optimum is **smooth** (linear feedback).

4. It generalizes to constraints, inequality constraints, free terminal time, and stochastic versions.

For VQAs, the PMP gives a non-obvious viewpoint on parameter optimization that is sometimes more natural than gradient descent.

The PMP Applied To VQAs

For a VQA, the optimal control problem is:

- **State:** $|\psi(t)\rangle$, evolving via discretized Schrödinger equation.
- **Control:** parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P)$, one per gate (or one per gate-time interval).
- **Dynamics:** unitary evolution under the gate Hamiltonians.
- **Cost:** terminal expectation $C = \langle\psi(T)|H|\psi(T)\rangle$, no running cost.

Discretized PMP for VQA:

The "time" is the position in the circuit. The state $|\psi_k\rangle$ at step k is the result of applying the first k gates. The costate $|\lambda_k\rangle$ is propagated *backward* from $|\lambda_T\rangle = H|\psi_T\rangle$ using the adjoint dynamics.

The **gradient with respect to** θ_k is then

$$\frac{\partial C}{\partial \theta_k} = -i\langle\lambda_k|\frac{\partial U_k}{\partial \theta_k}U_k^\dagger|\psi_k\rangle + \text{c.c.},$$

which is exactly the form one gets from the parameter-shift rule. **The PMP-derived gradient and the parameter-shift gradient are the same expression in different notation.**

Why this matters: the costate-based view shows that the gradient computation is doing **forward propagation of the state** plus **backward propagation of the costate**, with a contraction at each step. This is structurally identical to backpropagation in classical neural networks. The "VQA backprop" picture is the PMP applied to circuits.

This is not new — Schuld et al. and others have made the connection explicit — but the PMP formulation is the *most general* statement of the underlying structure.

Bang-Bang Control And QAOA

A particularly important consequence of the PMP: when the control Hamiltonian is **linear in the control variable**, the optimum is **bang-bang** — the control switches between its extreme values.

For QAOA: the ansatz is

$$|\psi(\boldsymbol{\beta},\boldsymbol{\gamma})\rangle = \prod_k e^{-i\beta_k H_M} e^{-i\gamma_k H_C} |+\rangle^{\otimes n},$$

where H_M is the mixer and H_C is the cost. This is a *piecewise-constant control*: at each step, the control selects either H_M or H_C, and the only choice is the duration.

PMP analysis. The control Hamiltonian for the optimal control problem is

$$H_{\text{ctrl}}(\boldsymbol{\theta}) = i\boldsymbol{\lambda}^\dagger \cdot (H_M\beta + H_C\gamma)|\psi\rangle + \text{c.c.}$$

This is *linear* in the durations β_k, γ_k. So the optimum durations are at the *boundary* of their allowed range — which is the bang-bang result.

Implication: the optimal QAOA (in the unconstrained-duration limit) uses only "extreme" values for β_k, γ_k — it switches abruptly between the cost and mixer Hamiltonians. Real QAOA has constrained durations (typically $[0, \pi]$), which softens the bang-bang prediction but the underlying structure is preserved.

This was first observed by Brandao et al. (2018) and subsequently studied as the **bang-bang QAOA** literature. The connection to Pontryagin gives the theoretical foundation.

Singular Arcs And QAOA Difficulty

A subtlety: when the control Hamiltonian is *exactly* zero in some region (the coefficient of the control vanishes identically), the bang-bang result fails — you have a **singular arc** where the optimal control is determined by higher-order conditions.

For VQAs: singular arcs correspond to **flat regions of the cost landscape** — the gradient (which is the linear coefficient of the control in the Hamiltonian) is identically zero. **Barren plateaus are exactly singular arcs in the optimal control formulation.**

This gives a *third* characterization of barren plateaus, complementary to: - Module 4's measure-theoretic view (Haar 2-design integration). - Module 8's Lie-algebraic view (small gradient variance from large DLA). - Now: **the plateau is a region where the PMP minimum condition is degenerate** — the control variable doesn't appear linearly in the Hamiltonian, so the standard gradient information is gone.

Implication for optimizer design: in a singular arc, you cannot use the PMP-derived gradient. You need either second-order conditions (Hessian) or non-gradient methods (population optimizers, Module 6.7-6.8). This is the optimal-control justification for Module 6's hybrid methods.

The Dual Variable Interpretation

A useful interpretation of the costate $\boldsymbol{\lambda}(t)$: it is the **dual variable** for the dynamics constraint. In Lagrangian terms, $\boldsymbol{\lambda}^T f(\mathbf{x}, \mathbf{u})$ is the penalty for violating $\dot{\mathbf{x}} = f$.

For VQAs, this means the costate $|\lambda_k\rangle$ is the "sensitivity of the final cost to perturbations of the state at step k." Knowing the costate at every step tells you exactly how much each intermediate quantum state matters for the final outcome.

Practical use: the costate identifies the *bottleneck* gates in a circuit — the gates whose perturbations have the largest effect on the final cost. These are the gates that should be parameterized most carefully and updated most aggressively. Gates with small costate norm are not contributing much and can be left alone or removed (this is the basis of **adaptive ansatz** methods like ADAPT-VQE).

Numerical Methods For PMP

Solving the PMP numerically is hard because it's a two-point boundary value problem (state at $t = 0$, costate at $t = T$).

Standard methods:

1. **Shooting.** Guess $\boldsymbol{\lambda}(0)$, integrate forward, check if the terminal condition is satisfied, adjust the guess.
2. **Multiple shooting.** Break the trajectory into segments, with a guess for the state and costate at each segment boundary.
3. **Collocation.** Discretize state and costate as polynomials over time intervals; solve the resulting algebraic system.
4. **Indirect methods generally.** Solve the PMP system directly as a system of nonlinear equations.

Compared to gradient descent. Gradient descent on the parameters is the *direct method* — discretize, then optimize — and is much simpler. The PMP-based methods are theoretically cleaner but numerically harder.

For most VQAs, gradient descent dominates. The PMP is used for *theoretical analysis* (proving things about optimal trajectories), not for the actual optimization.

When The PMP Gives New Information

A practical question: does the PMP ever give VQA practitioners *new* information they couldn't get from parameter-shift gradient descent?

Yes, in three specific situations:

1. Bang-bang structure prediction. Knowing that the optimum is bang-bang lets you initialize at the extreme values, which often gives better convergence than uniform initialization.

2. Singular arc detection. If you compute the costate norm and it's small, you're in a singular region (barren plateau). This is a fast diagnostic for plateaus.

3. Adaptive ansatz design. The costate identifies which gates contribute most. Adaptive ansatz methods (ADAPT-VQE) use this to grow the ansatz one gate at a time.

These are not earth-shattering, but they are concrete benefits. They justify the theoretical investment in understanding the PMP for VQA practitioners.

Stochastic And Quantum Pontryagin

A few extensions of the PMP relevant to VQAs:

1. Stochastic PMP. Generalizes to systems with noise, replacing the deterministic costate with a stochastic process. Used in stochastic optimal control. For VQAs with shot noise, this is the right framework, but the analysis is more complex.

2. Quantum PMP. Bonnard, Sugny, and others have developed PMP variants for quantum systems with explicit unitary constraints. These give time-optimal results for specific quantum gates (e.g., Khaneja-Brockett-Glaser).

3. Chattering (Fuller's phenomenon). In some optimal control problems, the optimal trajectory has *infinitely many* bang-bang switches in finite time. This is a pathological case but it appears in some quantum optimal control problems and is worth being aware of.

Structural Role

This node is the **theoretical capstone** of Module 7's optimal control thread. It introduces the Pontryagin minimum principle, applies it to VQAs, identifies the costate-gradient connection, and explains the bang-bang and singular-arc structures that appear in QAOA-like ansätze.

It directly connects to node 7.6 (feedback control), which uses the PMP framework to design closed-loop optimizer architectures.

Relations to Other Concepts

- **Pontryagin et al. (1962)** — *The Mathematical Theory of Optimal Processes*, the original.
- **Brandão et al. (2018)** — bang-bang QAOA.
- **Bonnard-Sugny** — quantum optimal control via PMP.
- **Module 6.1 (Gradient Descent)** — the practical use of PMP-derived gradients.
- **Module 4 (Barren Plateaus)** — singular arcs as plateaus.

Worked Example — PMP For A 1-Qubit State Preparation

Problem. Steer a 1-qubit state from $|0\rangle$ to $|1\rangle$ using the control Hamiltonian $H(t) = u(t)X$, with $|u(t)| \leq 1$.

Setup. - State: $|\psi(t)\rangle = c_0(t)|0\rangle + c_1(t)|1\rangle$. - Cost: terminal fidelity to $|1\rangle$, $J = -|c_1(T)|^2$. - Dynamics: $i|\dot{\psi}\rangle = u(t)X|\psi\rangle$.

PMP analysis. The control Hamiltonian is linear in u, so the optimum is bang-bang: $u^*(t) = \pm 1$.

Solving. With $u = +1$, the dynamics rotates the state at unit angular velocity around the X axis. To go from $|0\rangle$ to $|1\rangle$, you need to rotate by π. The minimum time is $T^* = \pi$.

Costate. The costate is $|\lambda(t)\rangle = \pm 2\text{Im}[c_1^*]|1\rangle - 2\text{Re}[c_1^*]|0\rangle$ or similar — geometrically, it's the gradient of $-|c_1|^2$ with respect to the state.

Connection to a single-parameter VQA. The minimum time π corresponds to the parameter $\theta = \pi$ in $R_X(\theta)$. The "bang-bang" optimum corresponds to "use the maximum allowed parameter value." This is trivial in this 1-parameter case but generalizes.

For multi-parameter ansätze, the bang-bang structure says: in each gate's parameter, the optimum tends to be at the extremes of the allowed range. This is *not* true in general for VQAs (because the cost is non-convex), but it is the right intuition in many cases.

Visualization

Pending. A diagram of a bang-bang control trajectory: the control $u(t)$ jumps between $+1$ and -1, the state evolves smoothly, the costate is computed by backward propagation. Caption: "Optimum, costate, control: PMP in three lines."

Exercises

1. For an LQR problem, derive the optimal feedback law from the PMP. How does it match the Riccati result?
2. The PMP says the costate satisfies $\dot{\boldsymbol{\lambda}} = -\partial H_{\text{ctrl}}/\partial \mathbf{x}$. For a VQA with terminal cost only, derive the costate dynamics.

3. (VQA) For a QAOA at depth $p = 2$ with a 4-vertex MaxCut, write the discretized PMP system. How many costate variables are there?

Research Extensions

- **PMP-aware VQA optimizers** — using costate information to drive parameter updates beyond simple gradient descent.
- **Singular arc detection in practice** — fast tests for whether you're in a barren plateau using costate norms.
- **Bang-bang VQA ansatz design** — explicitly constructing ansätze that have bang-bang optima for tractable problems.
- **Stochastic PMP for shot-noise VQAs** — formal extension to noisy gradient estimates.

Cross-links to Other Courses

- **Calculus of Variations** — Euler-Lagrange equations, Hamilton's principle.
- **Optimal Control (Lewis-Vrabie-Syrmos)** — the textbook treatment.
- **Quantum Optimal Control (D'Alessandro)** — the quantum-specific variant.
- **Module 7.4 (Optimal Control Theory)** — the parent framework.
- **Module 6 (Optimization Dynamics)** — the practical optimizer side.

Feedback Control in Quantum Systems

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Control-Theoretic View **Node:** 7.6

This is the **closing node** of Module 7. It is the synthesis: having developed observability (7.2), controllability (7.3), optimal control (7.4), and Pontryagin (7.5) as building blocks, this node combines them into the central object of modern control theory — the **closed-loop feedback controller**.

A feedback controller is a system that **observes** the state, **decides** what control input to apply based on the observation, and **acts**. The defining feature is that the controller's decisions depend on the observed state — closing the loop. This is what makes control systems robust to noise, drift, and disturbances; it is the technological basis of every modern engineered system from cruise control to power grids.

For VQAs, feedback control is the right framework for the *adaptive* extension. A non-adaptive VQA has a fixed optimizer schedule and ignores the system's response. An adaptive VQA observes the system (gradient norms, cost trajectory, hardware drift) and modifies its strategy accordingly. The control-theoretic view says: this is a feedback controller, and you should design it as one — using the same principles that have worked for 70 years in classical engineering.

This node walks through feedback control concepts, the quantum measurement complications, and the implications for adaptive VQA design. It is the natural bridge to Module 10 (VQA as Adaptive Control Systems).

What Feedback Control Is

A **feedback control system** has:

- A **plant** (the system being controlled).
- A **sensor** (measures the plant's output).
- A **controller** (decides the input based on the measurement).
- An **actuator** (applies the input to the plant).

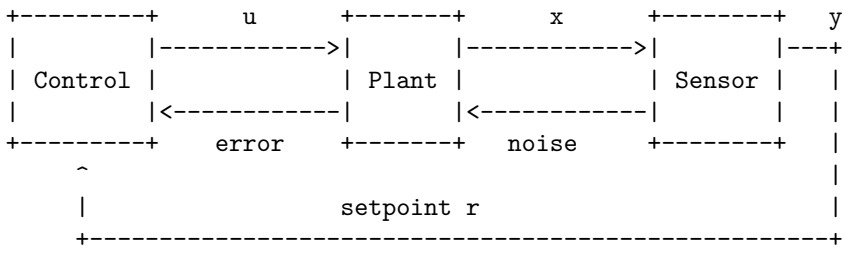
The signal flow is: plant → sensor → controller → actuator → plant. The loop is *closed* because the controller's output depends on the plant's measured output.

Compare to open-loop control: in open loop, the controller just runs a pre-computed input sequence and doesn't observe the plant. Open loop is simple but fragile — any disturbance or model error throws it off.

Closed loop is robust because errors get observed and corrected. This is why every real engineered system uses feedback.

The Standard Feedback Block Diagram

The canonical block diagram:



The controller compares the measured output y against a desired setpoint r, computes an error, and produces a control input u. The plant state x evolves under u, the sensor measures $y = h(x) + \text{noise}$, and the loop continues.

The **controller design problem** is: given the plant's dynamics and the sensor characteristics, design the controller logic so that the closed-loop system has desired properties (stability, fast convergence, disturbance rejection, etc.).

Classical Feedback Designs

A few standard patterns:

1. Proportional-Integral-Derivative (PID).

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau + K_d \cdot \dot{e}(t),$$

where $e(t) = r - y(t)$ is the error. Three tunable gains. Simple, robust, dominates classical industrial control. Most servos, thermostats, and chemical reactors use PID.

2. State feedback. $u = -K\mathbf{x}$ for a constant gain matrix K. Requires full state observation. Optimal under LQR assumptions.

3. Output feedback. $u = -K\mathbf{y}$ — directly from measurements without estimating the state. Simpler but less powerful than state feedback.

4. Observer-based feedback (separation principle). First estimate the state $\hat{\mathbf{x}}$ from measurements (Kalman filter), then apply state feedback to the estimate: $u = -K\hat{\mathbf{x}}$. Decouples state estimation from control design.

5. Model Predictive Control (MPC). At each time step, solve a finite-horizon optimal control problem online, apply only the first action, then repeat. Handles constraints naturally. Used in process control and (more recently) robotics.

6. Adaptive control. The controller's parameters update online as the plant is observed. Useful when the plant model is uncertain or slowly varying. Module 10 takes this up.

Why Quantum Feedback Is Different

Translating these to the quantum setting runs into the basic constraint of quantum measurement: **measurement collapses the state**.

This means:

1. You cannot continuously observe the state. Each measurement destroys the state (or at least disturbs it). To get a continuous trace of the state, you need many copies of the system or a *weak* measurement scheme.

2. The measurement is intrinsically noisy. Even an ideal projective measurement gives a sample from a probability distribution, not the underlying state directly.

3. The feedback delay matters. If the time between measurement and feedback action is non-negligible, the state has already evolved during that delay, and the action is based on stale information.

These constraints have spawned a whole sub-field: **quantum feedback control**. The two main paradigms:

(a) Discrete feedback. Measure projectively, then apply a unitary or reset based on the measurement outcome. Repeat. Used in error correction, state preparation, and discrete quantum computation.

(b) Continuous feedback. Use weak measurements that perturb the state minimally, integrate the measurement record to estimate the state in real time, and apply Hamiltonian feedback proportional to the estimate. The framework is **stochastic master equations** plus **conditional dynamics** — Wiseman-Milburn formalism.

For VQAs, the relevant paradigm is **discrete feedback at the optimizer level**: measure the cost (or gradient), update the parameters, then repeat. The "measurement" is the cost estimation, the "controller" is the optimizer, the "actuator" is the parameter update mechanism.

VQA As A Discrete Feedback Loop

The VQA outer loop is *exactly* a discrete feedback control loop:

Feedback element	VQA element
Plant	Quantum circuit + cost evaluation
Sensor	Cost estimator (with shot noise)
Controller	Optimizer (Adam, GD, HOPSO, etc.)
Actuator	Parameter update
Setpoint	Target cost (often $-\infty$ or known optimum)
Loop frequency	Iterations per second
Loop delay	Time per iteration (state prep + measurement + classical compute)

Implications of this framing:

1. **The optimizer is a controller.** It can be designed using control-theoretic principles, not just machine-learning intuitions.
2. **Loop delay matters.** If iterations are slow, the controller's actions are based on stale information. This affects optimizer choice — fast-converging optimizers (Adam) are better when iterations are slow.
3. **Sensor noise matters.** Shot noise in cost estimation is exactly sensor noise in the feedback loop. It can be filtered (Kalman-style), averaged (running mean), or compensated (Robbins-Monro).
4. **Stability is a concern.** In feedback loops, large gains can cause instability (oscillations, divergence). In VQAs, large learning rates have the same effect. The control framing tells you to think about gain scheduling and stability margins, not just convergence rates.
5. **Disturbance rejection is a concern.** Hardware drift acts as a disturbance on the plant. A good optimizer should reject this disturbance — adapt as the hardware changes. This is the adaptive control problem.

The Separation Principle For VQAs

Classical control theory has a beautiful result: under linearity and Gaussian noise, the optimal controller can be **separated** into:

1. A **state estimator** (Kalman filter) that produces a best-estimate state from the measurement record.
2. A **feedback law** (LQR-style) that acts on the estimate.

The two can be designed *independently* and the result is jointly optimal. This is the **separation principle**.

For VQAs, the natural analogue:

1. **State estimator:** an estimator that maintains a posterior over the parameter values (or the cost landscape) given the noisy cost evaluations.
2. **Feedback law:** an optimizer that acts on the posterior estimate.

A naive VQA optimizer skips the estimator and feeds raw measurements into the controller — equivalent to a "no-filter" controller. Adding even a simple moving-average filter (or a Kalman-like estimator) significantly improves performance under noise.

Bayesian VQA optimizers (Bayesian optimization, Gaussian process bandits) implement this separation explicitly: they maintain a posterior over the cost landscape and update parameters based on the posterior. They are the cleanest example of separation-principle thinking in VQAs.

Robust Control For VQAs

A more advanced control-theoretic concept: **robust control** handles uncertainty in the plant model. Classical methods include H_∞ control, μ-synthesis, and Lyapunov-based robust design.

For VQAs, the "uncertainty" comes from:

1. **Hardware drift** (gates calibrate slightly differently over time).
2. **Noise model uncertainty** (you don't know the exact noise rates).
3. **Cost evaluation noise** (shot noise, plus systematic errors).
4. **Optimizer hyperparameter uncertainty** (you don't know the right learning rate).

A robust VQA controller would be designed to **achieve good performance for all plants in some uncertainty set**. This is conservative but reliable — and it is the right framework when you can't trust your plant model perfectly.

Practical robust techniques for VQAs:

- **Conservative learning rates:** stay below the stability threshold for all plausible plants.
- **Trust regions:** limit the parameter step size so a single bad measurement doesn't move you far.
- **Worst-case cost evaluation:** average over multiple shots or multiple circuits to reduce noise sensitivity.
- **Adaptive robust controllers:** learn the plant uncertainty online and tighten the bounds as more data arrives.

These are not standard in the VQA literature but are routine in classical control. Importing them is one of the underexploited bridges between fields.

Adaptive Optimizers As Adaptive Controllers

The thesis's central claim: **adaptive VQA optimizers (HOPSO, Adam with gain scheduling, regularized SGD) are adaptive controllers**, and they should be designed as such.

The argument:

1. The standard fixed-hyperparameter optimizer is the equivalent of an open-loop controller — runs a pre-computed schedule, ignores system response.
2. An optimizer that observes its own performance (gradient norms, cost trajectory, gradient variance) and modifies its hyperparameters online is a closed-loop adaptive controller.
3. The mature literature on adaptive control gives explicit design principles, stability theorems (Lyapunov-based), and convergence guarantees that apply with appropriate translation.
4. Most VQA optimizers are designed by hand-tuning rather than by adaptive control principles. There is significant room for principled improvement.

This is the bridge to **Module 10**, which takes the adaptive control framing and develops it into specific architectures for VQA deployment.

Quantum Feedback Control: A Brief Survey

For completeness, the dominant approaches in *quantum* feedback control (the inner-loop, gate-level version):

1. Wiseman-Milburn continuous quantum measurement. Weak measurements with Wiener-process measurement records. Stochastic master equations describe the conditional dynamics. The optimal controller is a function of the measurement record.

2. Quantum error correction. Repeatedly measure stabilizers, decode the syndrome, apply correction. The discrete version of feedback. Surface codes, color codes, Bacon-Shor codes all use this paradigm.

3. Measurement-based quantum computation. The whole computation is measurement and feed-forward. Each measurement determines the next, and the final state encodes the result. Inverts the usual "prepare then measure" structure.

4. Quantum Kalman filtering. Uses Wiseman-Milburn formalism to maintain a real-time state estimate, then applies feedback to drive the estimate toward a target. Used in cooling experiments and quantum metrology.

These are not VQA techniques per se, but they show that quantum feedback control is a well-developed field with established methods. Importing them into VQA design (especially for hardware-aware optimizers) is an open research direction.

Closing Module 7

Module 7 has now traversed the control-theoretic view of VQAs:

- **7.1:** The framing — VQA as a control system, two-loop structure.
- **7.2:** Observability — what we can learn from measurements, gauge symmetries.
- **7.3:** Controllability — what states we can reach, the DLA equivalence with expressivity.
- **7.4:** Optimal control theory — the algorithmic core, GRAPE, Krotov, CRAB.
- **7.5:** Pontryagin minimum principle — the structural conditions for optimality.
- **7.6 (this node):** Feedback control — closing the loop, adaptive optimizers.

The arc: from passive structural analysis (observability, controllability) to active design (optimal control, PMP) to closed-loop adaptation (feedback). Each step adds a layer that the previous one lacked.

The thesis's central claim from this module: **VQAs are control systems, and 70 years of control theory has techniques the VQA community has not fully imported**. Modules 8 (theory) and 10 (deployment) build on this premise.

Structural Role

This node is the **closing synthesis** of Module 7. It combines observability, controllability, and optimal control into the closed-loop feedback framework, identifies VQA optimization as a discrete feedback loop, and previews the adaptive control deployment of Module 10.

It is also the strongest argument for the control-theoretic view: closed-loop adaptive controllers are the technology that makes modern engineering work, and importing them into VQA design is one of the best-leveraged research directions for making VQAs robust to real hardware.

Relations to Other Concepts

- **Wiener (1948)** — *Cybernetics*, founding text.
- **Kalman (1960)** — Kalman filter, separation principle.
- **Wiseman-Milburn (2010)** — *Quantum Measurement and Control*.
- **Module 6 (Optimization Dynamics)** — the optimizer-as-controller view.
- **Module 10 (Adaptive Control Systems)** — the deployment endpoint.

Worked Example — A Feedback VQA Optimizer

Consider a VQA optimizer that:

1. Estimates the cost $\hat{C}_k$ at iteration k.
2. Computes a moving-average filtered cost $\bar{C}_k = \alpha \bar{C}_{k-1} + (1-\alpha)\hat{C}_k$ (Kalman-like filter).
3. Uses $\bar{C}_k$ (not $\hat{C}_k$) to compute the gradient via parameter shift.
4. Adapts the learning rate based on the variance of recent gradients: low variance → larger step, high variance → smaller step.
5. Detects "stuck" behavior (cost not decreasing) and applies a random kick to the parameters.

As a feedback controller: - **Plant:** the quantum circuit + Hamiltonian. - **Sensor:** raw cost estimator (noisy). - **Filter:** the moving average — denoises the sensor. - **Controller:** the gradient-based parameter update with adaptive gain. - **Actuator:** parameter modification. - **Disturbance rejection:** the random kick when stuck.

This is *exactly* how a competent classical control engineer would design a controller for a noisy plant. None of the techniques are novel in control theory; all are well-established. But applied to VQAs, this kind of design is uncommon — most optimizers are hand-tuned with a single learning rate and no filtering.

The control framing motivates the use of these techniques and gives the principled justification.

Visualization

Pending. A block diagram of a closed-loop adaptive VQA optimizer, with all the control components labeled (plant, sensor, filter, controller, actuator, adaptation, disturbance rejection). Caption: "VQA optimization, redrawn as classical control."

Exercises

1. For a VQA optimizer with cost noise std σ, what is the equivalent "sensor noise" in a classical feedback loop? How would a Kalman filter improve it?
2. The separation principle says estimator and controller can be designed independently under LQG assumptions. For a VQA, what would the "estimator" be?
3. (VQA) Design a feedback controller for a VQA optimizer that handles slow hardware drift (gate fidelities changing on the hour timescale). Specify the sensor, filter, controller logic.

Research Extensions

- **Kalman-filtered VQA optimizers** — explicit Kalman filtering of the cost trajectory.
- **Robust control for hardware-noise VQAs** — applying H_∞ techniques to optimizer design.
- **MPC-style VQA optimizers** — finite-horizon lookahead with constraint handling.

- **Quantum feedback at the gate level for VQAs** — combining inner-loop quantum feedback with outer-loop optimization.

Cross-links to Other Courses

- **Classical Control (Ogata)** — PID, state feedback, observer-based feedback.
- **Quantum Measurement and Control (Wiseman-Milburn)** — quantum-specific feedback theory.
- **Adaptive Control (Åström-Wittenmark)** — the parent literature for Module 10.
- **Module 6 (Optimization Dynamics)** — the optimizer-only view.
- **Module 10 (Adaptive Control Systems)** — the deployment chapter.

Module 8 — Expressivity Vs Trainability

Expressivity Theory

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.1

This is the **opening node** of Module 8 — the module that develops, in quantitative form, the central tradeoff that has been hovering implicitly behind every module since Module 4: **the more states an ansatz can reach, the harder it is to optimize over**.

Before we can talk about the tradeoff, we need to define **expressivity** with enough precision to be useful. That is the job of this node and node 8.2.

The intuitive picture: an ansatz $U(\boldsymbol{\theta})$ defines a *family* of quantum states $\{|\psi(\boldsymbol{\theta})\rangle\}$ as $\boldsymbol{\theta}$ ranges over its domain. Expressivity is "how big the family is" — formally, how much of the Hilbert space the family covers, in some appropriate measure.

This node introduces the formal definitions, explains the multiple ways expressivity can be measured, and sets up the trainability counterpart in node 8.4.

What "Expressivity" Means

Several closely-related but distinct definitions appear in the literature:

1. Reachability: The set of states $\mathcal{R}(U) = \{|\psi(\boldsymbol{\theta})\rangle : \boldsymbol{\theta} \in \Theta\}$. The "reachable manifold" of the ansatz. Expressivity is, intuitively, the size of $\mathcal{R}(U)$.

2. Volume: For an ansatz with P parameters, the manifold $\mathcal{R}(U)$ has dimension at most P. The "volume" of $\mathcal{R}(U)$ (in the Fubini-Study metric) is a measure of how much of the state space it covers.

3. Distribution distance: Compare the distribution induced on states by random parameter values to the Haar measure on the full state space. An ansatz is "highly expressive" if these distributions are close, "low expressivity" if the induced distribution is far from Haar.

4. Approximation error: For a given target state $|\phi\rangle$, the minimum distance from $|\phi\rangle$ to any state in $\mathcal{R}(U)$. A low approximation error (for any target) means high expressivity.

5. Entanglement coverage: The range of entanglement entropies that states in $\mathcal{R}(U)$ can achieve. Higher coverage = higher expressivity (in the entanglement sense).

These definitions are not equivalent. An ansatz can be highly expressive in one sense and not in another. The thesis (and most recent VQA literature) treats expressivity as a *family* of related but distinct quantities, each useful for different purposes.

Why Expressivity Matters

Expressivity matters for two reasons that pull in opposite directions:

1. You need enough expressivity to represent the answer. If your target state isn't in the reachable manifold of your ansatz, no amount of optimization will find it. The ansatz must be expressive enough that the ground state (or a good approximation to it) lies in $\mathcal{R}(U)$.

2. You need expressivity to be limited or you'll hit a barren plateau. An ansatz that can reach all of Hilbert space is, by Module 4's analysis, exactly the ansatz that has barren plateaus. Too much expressivity is fatal for trainability.

The right amount of expressivity is "just enough." Enough to contain the answer, no more. Designing an ansatz to hit this sweet spot is what Module 2's discussion of problem-inspired ansätze (UCCSD, HVA, ADAPT-VQE) was trying to achieve.

Reachable Manifold Dimension

The simplest expressivity measure is the **dimension** of the reachable manifold.

For a parameterized circuit with P parameters, the manifold $\mathcal{R}(U)$ has dimension at most P. It can be **less** than P if there are gauge redundancies (Section 3.4) or if some parameter combinations are degenerate.

For example, the ansatz $R_X(\theta_1)R_X(\theta_2)|0\rangle$ has 2 parameters but dimension 1 (because only $\theta_1 + \theta_2$ matters).

The full state space of n qubits is a manifold of dimension $2 \cdot 2^n - 2$ (real coordinates on the projective Hilbert space). For an ansatz to be "fully expressive" in the dimension sense, it needs at least this many parameters.

For $n = 4$, that's 30 parameters minimum. For $n = 10$, 2046. For $n = 20$, about 2 million. **Truly universal ansätze are exponentially expensive in parameter count.**

In practice, useful ansätze have parameter counts polynomial in n and reach a polynomially-large submanifold of the exponentially-large state space. This "polynomial expressivity" regime is where almost all VQA work lives.

The Sim-Johnson-Killoran Expressibility

The most widely-used quantitative expressivity measure for VQA was introduced by **Sim, Johnson, Killoran (2019)**. The idea: compute the distribution of fidelities between pairs of random ansatz states, and compare to the Haar distribution.

Let $P_{\text{Haar}}(F)$ be the Haar distribution of $F = |\langle\psi_1|\psi_2\rangle|^2$ for two Haar-random states. This has known closed form: $P_{\text{Haar}}(F) = (d-1)(1-F)^{d-2}$ where $d = 2^n$.

Let $P_{\text{ansatz}}(F)$ be the analogous distribution for two random ansatz states $|\psi(\boldsymbol{\theta}_1)\rangle, |\psi(\boldsymbol{\theta}_2)\rangle$ with $\boldsymbol{\theta}_i$ drawn uniformly.

Expressibility is the KL divergence:

$$\text{Expr} = D_{KL}\big(P_{\text{ansatz}}(F) \parallel P_{\text{Haar}}(F)\big).$$

Lower KL divergence = ansatz distribution is closer to Haar = "more expressive." Higher KL divergence = ansatz is structured = "less expressive."

This is the **standard expressivity metric** in the VQA literature. Most ansatz comparison papers report it.

Practical computation: sample many parameter pairs, compute the empirical fidelity distribution, and bin it. Compute the KL divergence to the analytic Haar distribution.

Expressivity Of Common Ansätze

The Sim-Johnson-Killoran metric has been computed for many standard ansätze. Some representative results (for $n = 4$ qubits):

- **Hardware-Efficient Ansatz, deep:** expressibility close to 0 (Haar-like) — fully expressive.
- **Hardware-Efficient Ansatz, 1 layer:** expressibility ~ 0.1 — limited but nontrivial.
- **UCCSD:** expressibility ~ 0.5 — strongly restricted by particle-number conservation.

- **Tensor Network ansatz (MPS):** expressibility depends on bond dimension; for low bond dimension, very limited.
- **Random Pauli circuit, depth 1:** expressibility ~ 1.0 — almost no coverage.

The trend: **as you add structure (symmetry, sparsity, restricted connectivity), expressibility decreases**. As you remove structure (random gates, deep circuits, generic parameterization), it increases.

This is the operational measurement of the tradeoff Module 4 discussed qualitatively.

Expressivity And Entanglement

Expressivity correlates with — but is not the same as — entanglement coverage.

A high-expressivity ansatz typically reaches highly entangled states (because Haar-random states are highly entangled, by Page's theorem from Section 4.3). A low-expressivity ansatz typically reaches a restricted entanglement range.

But the relationship is not exact:

- A product-state ansatz has zero entanglement and zero expressivity.
- A maximally-entangled (Bell-state-only) ansatz has high entanglement but low expressivity (small reachable set).
- A Haar-random ansatz has high entanglement and high expressivity.

The two axes are related but distinct. Module 4.3 framed barren plateaus in terms of entanglement; this module frames them in terms of expressivity. Both are valid; they capture overlapping but not identical structure.

For practical purposes, **entanglement-based reasoning** (Module 4.3) is more physical and gives you mitigation strategies; **expressivity-based reasoning** (this module) is more mathematical and gives you metrics for ansatz comparison.

Lie Algebraic Expressivity

A more recent and theoretically clean view: expressivity is governed by the **dynamical Lie algebra** of the ansatz generators.

Let $U(\boldsymbol{\theta}) = e^{-i\theta_L H_L} \cdots e^{-i\theta_1 H_1}$ for Hermitian generators $H_1, \ldots, H_L$. The dynamical Lie algebra (DLA) $\mathfrak{g}$ is the smallest Lie algebra containing all the H_j and closed under commutators $[H_i, H_j]$.

Theorem: the reachable manifold $\mathcal{R}(U)$ is contained in the orbit $\{e^{iG}|0\rangle : G \in \mathfrak{g}\}$. The dimension of this orbit is at most the dimension of $\mathfrak{g}$.

Implications: - Small DLA $\rightarrow$ small reachable manifold $\rightarrow$ low expressivity $\rightarrow$ high trainability. - Large DLA $\rightarrow$ large reachable manifold $\rightarrow$ high expressivity $\rightarrow$ low trainability.

For example: a UCCSD ansatz has a DLA equal to the (small) algebra of particle-number-conserving operators. A deep HEA has a DLA equal to the full $\mathfrak{u}(2^n)$, the Lie algebra of all n-qubit unitaries — exponentially large.

Ragone et al. (2024) developed a Lie-algebraic theory of barren plateaus that uses this DLA structure to give precise expressivity bounds. The DLA is the cleanest theoretical proxy for "how expressive is this ansatz" available in 2026.

Structural Role

This node is the **definitions** node of Module 8. It introduces the multiple senses of "expressivity" and previews the trainability counterpart in node 8.4.

The metric definitions in 8.2 and the Lie-algebraic view sketched here are the two technical foundations Module 8 will use throughout.

Relations to Other Concepts

- **Sim-Johnson-Killoran (2019)** — the standard expressibility metric.
- **Ragone et al. (2024)** — the Lie-algebraic theory.
- **Page's theorem** — Haar-random state entanglement.
- **Module 2 (Ansatz design)** — the practical context for the theory.
- **Module 4 (Barren plateaus)** — the trainability obstacle expressivity creates.

Worked Example — Counting Reachable Dimensions

Consider $|\psi(\theta_1, \theta_2)\rangle = R_Y(\theta_1)R_X(\theta_2)|0\rangle$ on 1 qubit.

The state space of 1 qubit is the Bloch sphere S^2, dimension 2. The ansatz has 2 parameters.

Compute reachability: $R_X(\theta_2)|0\rangle = \cos(\theta_2/2)|0\rangle - i\sin(\theta_2/2)|1\rangle$. Then $R_Y(\theta_1)$ rotates this around the Y-axis.

By inspection, the ansatz can reach every state on the Bloch sphere — it is universal for 1 qubit. Reachable manifold has dimension 2.

For 2 qubits, the state space has dimension 6. A 2-qubit ansatz with only single-qubit rotations $R_Y(\theta_1)\otimes R_Y(\theta_2)$ can only reach product states — a 2-dimensional submanifold of the 6-dimensional state space. Adding a CNOT after the rotations expands this to a 4-dimensional submanifold (entangled states reachable from product states by one CNOT). Adding more layers expands further, eventually reaching the full 6-dimensional manifold.

This dimension counting is the simplest expressivity calculation. The Sim-Johnson-Killoran metric, the Lie-algebraic dimension, and other measures all generalize this idea to higher-dimensional state spaces.

Visualization

Pending. A 2D projection of the Bloch sphere with reachable regions of three different ansätze highlighted: (a) a product-only ansatz (small dot), (b) a 1-layer HEA (larger region), (c) a deep HEA (the entire sphere). Caption: "Expressivity is the volume the ansatz reaches."

Exercises

1. For $|\psi(\theta)\rangle = R_Y(\theta)|0\rangle$ (1 qubit, 1 parameter), what is the reachable manifold? What is its dimension?
2. For an ansatz with 5 parameters in 4 qubits (state space dimension 30), what is the maximum possible reachable manifold dimension? Is the ansatz universal?
3. (VQA) The Sim-Johnson-Killoran expressibility for a deep HEA approaches the Haar value. Show analytically that for a 1-qubit ansatz $R_Y(\theta_1)R_X(\theta_2)R_Y(\theta_3)$, the expressibility is exactly Haar. (Hint: this is universal for 1 qubit.)

Research Extensions

- **Tighter expressivity-trainability bounds** — pinning down the exact constants in the tradeoff.
- **Problem-aware expressivity** — measures that account for the specific Hamiltonian, not just the ansatz.
- **Computational expressivity** — relating expressivity to computational hardness of the optimization.
- **Expressivity under noise** — how hardware noise modifies the effective expressivity of an ansatz.

Cross-links to Other Courses

- **Lie Theory** — Lie algebras, Lie groups, dynamical Lie algebras of control systems.
- **Differential Geometry** — manifolds, dimension, embedding.
- **Statistical Learning Theory** — VC dimension, capacity, generalization (the classical analogue of expressivity).
- **Module 2 (Parameterized Circuits)** — the operational ansatz design.
- **Module 4 (Barren Plateaus)** — the trainability obstacle.

Expressibility Metrics

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.2

Node 8.1 introduced expressivity as a *family* of related but inequivalent quantities. This node makes those quantities concrete: it walks through the **specific metrics** in the literature, how each one is computed, what each one tells you, and when each is the right tool.

The pragmatic motivation: if you are designing or comparing ansätze, you need a number — preferably one you can actually compute on a laptop. The pure theoretical view (reachable manifold dimension, Lie algebra dimension) doesn't scale. The metrics in this node **do** scale, at the cost of capturing only a slice of the full picture.

What A Good Expressibility Metric Should Do

Before walking through the metrics, the desiderata:

1. Computability. You should be able to compute it from samples (parameter draws + cost evaluations or fidelity calculations). No exact integrals, no exponential scaling.

2. Comparability. Two ansätze with the same number of parameters and qubits should give comparable numbers. The metric shouldn't be dominated by ansatz size.

3. Sensitivity. It should distinguish ansätze that differ in expressivity, not just in trivial ways.

4. Interpretability. A user should be able to read the number and know what it means qualitatively ("more expressive" or "less expressive").

5. Connection to trainability. Ideally, the metric correlates with trainability outcomes — knowing the number should help you predict whether the ansatz will train well.

No single metric meets all five. Each one trades off some desiderata for others. The literature has converged on a small handful that are "good enough" for most purposes.

Metric 1: Sim-Johnson-Killoran Expressibility

Already introduced in 8.1 — included here for completeness with the computational details.

Definition. Let $P_{\text{ansatz}}(F)$ be the distribution of fidelities $F = |\langle\psi(\boldsymbol{\theta}_1)|\psi(\boldsymbol{\theta}_2)\rangle|^2$ over uniformly sampled parameter pairs. Let $P_{\text{Haar}}(F) = (d-1)(1-F)^{d-2}$ for $d = 2^n$. Then

$$\text{Expr}_{\text{SJK}} = D_{KL}(P_{\text{ansatz}} \,\|\, P_{\text{Haar}}).$$

Computational protocol: 1. Sample M parameter pairs $(\boldsymbol{\theta}_1^{(m)}, \boldsymbol{\theta}_2^{(m)})$. 2. For each, compute $F^{(m)} = |\langle\psi(\boldsymbol{\theta}_1^{(m)})|\psi(\boldsymbol{\theta}_2^{(m)})\rangle|^2$. 3. Bin the fidelities into B bins to get an empirical distribution $\hat{P}_{\text{ansatz}}$. 4. Compute the KL divergence to the analytic Haar distribution.

Cost: M state preparation pairs, each requiring fidelity computation (which is $O(\text{poly}(n))$ on a simulator and harder on hardware — usually estimated via SWAP test or inversion test).

Interpretation: - $\text{Expr}_{\text{SJK}} \to 0$: ansatz fidelity distribution matches Haar — fully expressive (Haar-like coverage). - Expr_{SJK} large: ansatz is structured, far from Haar.

Strengths: computable, well-studied, comparable across ansätze with the same n. **Weaknesses:** dominated by global structure; misses fine-grained expressivity differences. The KL is sensitive to bin width and to the number of samples.

Metric 2: Entangling Capability

Also from Sim-Johnson-Killoran (2019). Measures the **average entanglement** of states produced by the ansatz.

Definition. Let $Q(|\psi\rangle)$ be the **Meyer-Wallach Q-measure**:

$$Q(|\psi\rangle) = 2\left(1 - \frac{1}{n}\sum_{k=1}^{n} \mathrm{Tr}(\rho_k^2)\right),$$

where ρ_k is the reduced density matrix on qubit k. $Q = 0$ for product states, $Q = 1$ for maximally entangled (in a specific sense). Then

$$\mathrm{Ent}(U) = \mathbb{E}_{\boldsymbol{\theta}}[Q(|\psi(\boldsymbol{\theta})\rangle)].$$

Computational protocol: sample parameters, prepare states, compute Q for each, average.

Cost: M state preparations and $O(n)$ reduced density matrices each.

Interpretation: - Ent ≈ 0: the ansatz produces near-product states. Limited expressivity. - Ent ≈ 1: the ansatz produces highly entangled states. - Haar value: roughly $1 - 1/(2^n + 1) \approx 1$ for large n.

Strengths: physically meaningful, easy to compute, complementary to SJK expressibility. **Weaknesses:** entanglement is a *consequence* of expressivity, not the same thing — high entanglement does not imply high reachability.

A good practice is to report **both** SJK expressibility and entangling capability, since they measure different facets.

Metric 3: Dynamical Lie Algebra Dimension

Already sketched in 8.1. A theoretically clean but computationally intensive metric.

Definition. Given the ansatz $U(\boldsymbol{\theta}) = \prod_j e^{-i\theta_j H_j}$, construct the dynamical Lie algebra $\mathfrak{g}$ by closing $\{H_j\}$ under commutators. Then

$$\mathrm{Expr}_{\mathrm{DLA}}(U) = \dim \mathfrak{g}.$$

Computational protocol: symbolic Lie algebra closure. Start with the generators, compute pairwise commutators, add new linearly independent operators, repeat until closure.

Cost: $O(\dim \mathfrak{g}^2)$ commutator computations and linear-independence checks. For an ansatz where $\dim \mathfrak{g}$ is polynomial in n, this is feasible. For deep ansätze where $\dim \mathfrak{g}$ is exponential, intractable.

Interpretation: - Small DLA dimension → small reachable manifold → low expressivity. - DLA dimension polynomial in n → "polynomially expressive" — typical regime for problem-inspired ansätze. - DLA dimension exponential in n → "fully expressive" — typical for HEA at sufficient depth.

Strengths: rigorous, ansatz-intrinsic (doesn't depend on parameter sampling), connects directly to trainability via the Ragone et al. theory. **Weaknesses:** scales poorly with depth; doesn't capture continuous expressivity differences (it's a single integer).

Use DLA dimension for theoretical analysis; use SJK expressibility for empirical comparison.

Metric 4: Effective Dimension

Introduced by Abbas et al. (2021) — "The power of quantum neural networks."

The idea: measure the **information geometric dimension** of the parameter space, weighted by the local geometry of the cost landscape.

Definition. Let $I(\boldsymbol{\theta})$ be the Fisher information matrix at $\boldsymbol{\theta}$. The effective dimension at scale γ is

$$d_{\text{eff}}(\gamma) = \frac{2\log\left(\frac{1}{V_\Theta}\int_\Theta \sqrt{\det(I_d + \gamma I(\boldsymbol{\theta}))}\, d\boldsymbol{\theta}\right)}{\log(\gamma/(2\pi\log n))},$$

where I_d is the identity, V_Θ is the parameter-space volume, and γ is a scale parameter. (Don't be intimidated — this is essentially "average rank of the Fisher information.")

Interpretation: - d_{eff} close to P (number of parameters): all parameters are independently informative — fully effective ansatz. - $d_{\text{eff}} \ll P$: many parameters are redundant — the effective parameter space is much smaller than P.

Strengths: captures redundancy, connects to generalization theory. **Weaknesses:** computing the Fisher information is expensive; the metric is defined at a scale γ and depends on it.

Effective dimension is the right metric when you suspect your ansatz has many redundant parameters (gauge degrees of freedom, near-degenerate directions). For most problem-inspired ansätze, it's overkill.

Metric 5: Frame Potential

A more theoretical metric: the **frame potential**.

Definition. For an ansatz $U(\boldsymbol{\theta})$ inducing a distribution on unitaries, the t-th frame potential is

$$\mathcal{F}_t(U) = \mathbb{E}_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}\left[|\mathrm{Tr}(U(\boldsymbol{\theta}_1)U(\boldsymbol{\theta}_2)^\dagger)|^{2t}\right].$$

For Haar-random unitaries, the frame potential takes a specific value depending on t and $d = 2^n$. An ansatz forms a **t-design** if its frame potential matches the Haar value:

$$\mathcal{F}_t(\text{ansatz}) = \mathcal{F}_t(\text{Haar}).$$

Interpretation: - The ansatz is a 1-design if its first moment matches Haar. - A 2-design if both first and second moments match. - More expressivity = higher t-design.

Strengths: rigorous, connects to the proof of barren plateaus (which assumes 2-designs). **Weaknesses:** needs to be computed at each order t; only practical for low t.

The frame potential is the *right* metric for connecting to barren plateau theory (Module 4 used it implicitly via "approximate 2-design"). For practical comparison of two ansätze, SJK or entangling capability is easier.

Metric 6: Approximation Error On A Test Suite

A purely operational metric. Define a set of "test states" $\{|\phi_k\rangle\}_{k=1}^{K}$. For each, compute the minimum error over the ansatz:

$$\epsilon_k = \min_{\boldsymbol{\theta}} \| |\phi_k\rangle - |\psi(\boldsymbol{\theta})\rangle \|.$$

Average over the test suite:

$$\text{Expr}_{\text{approx}} = \frac{1}{K} \sum_{k=1}^{K} \epsilon_k.$$

Interpretation: - Low average error → ansatz can approximate diverse target states → high expressivity. - High average error → ansatz misses many targets → low expressivity.

Strengths: directly operational — measures what you actually care about (can the ansatz represent my target?). **Weaknesses:** depends on the test suite. A test suite biased toward your ansatz's structure will give an unfair advantage.

Use this when you have a specific application in mind and a representative set of target states for that application.

Comparing The Metrics

A summary table:

Metric	Cost	Output range	Captures	Best for
SJK Expressibility	Moderate (samples)	KL divergence (positive)	Distributional coverage	General ansatz comparison
Entangling Capability	Cheap	$[0, 1]$	Entanglement structure	Quick check
DLA Dimension	Expensive (symbolic)	Integer	Reachable manifold dim	Theoretical analysis
Effective Dimension	Expensive	$[0, P]$	Parameter redundancy	Over-parameterized ansätze
Frame Potential	Moderate	Real number	t-design property	Plateau theory
Approximation Error	Application-dep.	Real number	Target representability	Application-driven

For most VQA work, the practical recommendation is: report **SJK expressibility + entangling capability** as the standard pair. Add **DLA dimension** if you can compute it. Add **effective dimension** if your ansatz has visible redundancy.

A Worked Comparison

Consider two 4-qubit ansätze:

Ansatz A: Hardware-Efficient, 2 layers. Single-qubit R_X, R_Y, R_Z on each qubit, then nearest-neighbor CNOTs. Repeat. Total: 24 parameters.

Ansatz B: UCCSD with 4 electrons. Trotterized particle-number-conserving excitations. Total: ~12 parameters.

SJK Expressibility: - A: ≈ 0.05 (close to Haar, highly expressive). - B: ≈ 0.5 (far from Haar — strongly restricted by particle-number conservation).

Entangling Capability: - A: ≈ 0.85 (nearly Haar entanglement). - B: ≈ 0.4 (limited entanglement — restricted to particle-number subspace).

DLA Dimension: - A: full $\mathfrak{u}(2^4) = 256$ — exponential in n. - B: $\binom{8}{4}^2 = 4900$ but restricted to particle-number sector, so $\binom{8}{4} = 70$.

Effective Dimension at $\gamma = 1$: - A: ≈ 18 (out of 24, some redundancy from gauge freedoms). - B: ≈ 11 (out of 12, near-fully effective).

Approximation Error on the H_2 ground state: - A: ≈ 0.001 (excellent — can represent the ground state). - B: ≈ 0.0001 (even better — UCCSD is *designed* for chemistry).

Conclusion: Ansatz A is more expressive in every general sense, but Ansatz B is *more expressive on the actual problem* because its restricted form happens to align with the chemistry symmetries. Both can solve H_2; they just use very different mechanisms.

Caveats On Metric Interpretation

1. Metrics measure structure, not performance. A more expressive ansatz isn't always better. As Module 4 showed, expressivity correlates with trainability obstacles. The right amount is "just enough."

2. Metric values depend on the parameter range. SJK expressibility, entangling capability, and effective dimension all depend on the parameter sampling distribution. Default is uniform on $[0, 2\pi]^P$, but if you constrain parameters (e.g., $[-\pi/4, \pi/4]$), the metrics change.

3. Sample noise. Empirical metrics (SJK, entangling capability) have sampling noise that scales as $1/\sqrt{M}$. For meaningful comparison, you need M large enough to resolve the difference between ansätze.

4. Comparability across qubit count. SJK expressibility and DLA dimension scale differently with n, so comparing ansätze on different qubit counts requires care. Within a fixed n, comparison is meaningful.

5. No metric captures cost-landscape geometry. None of the standard metrics tells you about local minima, barren plateaus, or convergence rates. They capture *static* expressivity, not *dynamic* trainability.

For trainability-aware ansatz comparison, you need the metrics from node 8.4 in addition.

Structural Role

This node is the **operational toolkit** of Module 8. It enumerates the actual numbers you compute and report when designing or analyzing an ansatz, with practical guidance on which metrics to use for which purpose.

It is the bridge between the abstract definitions of 8.1 and the trainability bounds of 8.4.

Relations to Other Concepts

- **Sim-Johnson-Killoran (2019)** — the original SJK metric paper.
- **Abbas et al. (2021)** — effective dimension.
- **Meyer-Wallach Q-measure (2002)** — the entanglement metric.

- **Frame potentials (Scott 2008)** — t-design theory.
- **Ragone et al. (2024)** — DLA expressivity.

Worked Example — Computing SJK Expressibility For A Toy Ansatz

Consider the 2-qubit ansatz $U(\theta_1, \theta_2) = R_Y(\theta_1) \otimes R_Y(\theta_2)$ — pure product, no entanglement.

Step 1: Sample parameter pairs. Draw $M = 1000$ pairs of (θ_1, θ_2) and (θ_1', θ_2') uniformly from $[0, 2\pi]^2$.

Step 2: Compute fidelities. $|\langle\psi(\theta_1, \theta_2)|\psi(\theta_1', \theta_2')\rangle|^2 = |\langle 0|R_Y^\dagger(\theta_1)R_Y(\theta_1')|0\rangle|^2 \cdot |\langle 0|R_Y^\dagger(\theta_2)R_Y(\theta_2')|0\rangle|^2$.

Each factor is $\cos^2((\theta_1' - \theta_1)/2)$, uniformly distributed in $[0, 1]$. The product is the product of two independent uniforms — distribution is $-\log(F)$ (concentrated near $F = 0$, heavy near 1).

Step 3: Compare to Haar. For $d = 4$, $P_{\text{Haar}}(F) = 3(1 - F)^2$. The ansatz distribution is dominated by small fidelities (most product states are nearly orthogonal), but the *concentration profile* is very different from Haar.

Step 4: Compute KL. The KL divergence is large (~ 1.5 in natural log units), reflecting that this ansatz is *far* from Haar.

For comparison, a 2-qubit ansatz with one CNOT after the rotations has SJK expressibility ≈ 0.3 — much closer to Haar but still structured.

A deep HEA on 2 qubits has SJK expressibility ≈ 0.01 — essentially Haar.

This worked example shows the kind of empirical comparison that drives ansatz design in practice.

Visualization

Pending. A bar chart of SJK expressibility for 6 standard ansätze (HEA-1, HEA-3, HEA-deep, UCCSD, MPS-2, MPS-8) on 4 qubits, with the Haar value marked as a horizontal line. Caption: "Lower bars are closer to Haar — more expressive."

Exercises

1. Compute the entangling capability of $U(\theta) = R_Y(\theta) \otimes R_Y(\theta)$ (2 qubits, 1 parameter, no entangling gate). What is it?
2. The SJK expressibility for an ansatz that produces only the state $|+\rangle^{\otimes n}$ regardless of parameters is what? (Hint: the fidelity distribution is a delta at 1.)
3. (VQA) For a 6-qubit HEA with 5 layers and a UCCSD ansatz with 6 electrons, predict the relative ordering of (a) SJK expressibility, (b) entangling capability, (c) DLA dimension. State any assumptions.

Research Extensions

- **Trainability-aware expressibility metrics** — metrics that incorporate gradient variance, not just static structure.
- **Application-conditioned expressibility** — metrics defined relative to a specific Hamiltonian or target distribution.
- **Hierarchical metrics** — multi-resolution expressibility (local vs global).
- **Sample-efficient expressibility estimation** — getting useful numbers from few samples.

Cross-links to Other Courses

- **Information Geometry** — Fisher information, effective dimension.
- **Random Matrix Theory** — Haar measure, frame potentials.
- **Module 4 (Barren Plateaus)** — the trainability cost of high expressibility.
- **Module 2 (Ansatz Design)** — the practical context.

Universal Approximation in Quantum Circuits

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.3

The classical neural network literature has a famous theorem: **a feedforward network with one hidden layer of sufficient width can approximate any continuous function** (Cybenko 1989, Hornik 1991). This is the *universal approximation theorem* and it is what justifies neural networks at the foundational level.

Quantum variational circuits have analogous universality results, and this node walks through them. The key questions:

1. **Is there a quantum analogue of universal approximation?** (Yes, several.)
2. **What does "universal" mean in the quantum setting?** (More than one thing.)
3. **What does universality cost in terms of circuit depth, parameter count, etc.?** (Exponentially much, in general.)
4. **How does universality relate to barren plateaus?** (Negatively — universal ansätze plateau.)

The punchline: quantum universal approximation theorems exist, but they are *existence statements* about ansätze that are exponentially deep or exponentially wide. The practically useful ansätze are not universal — they are restricted, problem-specific, and that restriction is what makes them trainable.

What "Universal" Could Mean For A Quantum Circuit

There are multiple senses in which a parameterized circuit could be "universal."

1. State universality. The ansatz can prepare any state in the Hilbert space:

$$\forall |\phi\rangle \in \mathcal{H},\ \exists \boldsymbol{\theta} : |\psi(\boldsymbol{\theta})\rangle = |\phi\rangle.$$

2. Unitary universality. The ansatz can implement any unitary on the Hilbert space:

$$\forall U \in U(2^n),\ \exists \boldsymbol{\theta} : V(\boldsymbol{\theta}) = U.$$

State universality is weaker (you only need to reach states starting from $|0\rangle^{\otimes n}$). Unitary universality is stronger (you need to be able to act on arbitrary inputs).

3. Function approximation universality. The map $\boldsymbol{\theta} \to \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ (or similar observable expectations) can approximate any continuous function on the parameter space. This is the closest analogue of classical universal approximation.

4. Distribution universality. The ansatz can sample from any distribution over computational basis states. Relevant for quantum generative models.

5. Operator universality. The ansatz can prepare any Hermitian observable as a linear combination of its parameters. Relevant for quantum machine learning kernels.

These are *not* equivalent. A circuit can be universal in one sense and not in another. The literature usually means **state universality** unless otherwise specified.

State Universality: Existence

Theorem (state universality). A parameterized circuit on n qubits with at least $2 \cdot 2^n - 2$ real parameters can be state-universal — that is, there exists a circuit construction with this many parameters that can reach every state in the projective Hilbert space.

Proof idea. The projective Hilbert space $\mathbb{CP}^{2^n-1}$ has real dimension $2 \cdot 2^n - 2$. A smooth, surjective map from a parameter space to this manifold must have parameter space dimension at least $2 \cdot 2^n - 2$. With this many parameters, an explicit construction (e.g., the **Schmidt decomposition based circuit** of Iten et al. 2016) achieves universality.

Cost: $2 \cdot 2^n - 2$ parameters means **exponential growth in qubit count**. For $n = 10$, 2046 parameters; for $n = 20$, 2 million; for $n = 50$, 10^{15}.

State universality is theoretically important but **practically infeasible** for any $n > 10$ or so. Universal ansätze are not used in practice; they are reference points to anchor the theory.

Unitary Universality: Existence

Theorem (unitary universality). A parameterized circuit on n qubits can implement any unitary in $U(2^n)$ if and only if its parameter count is at least $(2^n)^2 - 1 = 4^n - 1$.

Cost: $4^n - 1$ parameters — even worse than state universality. Doubly exponential in qubit count.

Unitary universality is **even more theoretical** than state universality. The only reason it's worth knowing about is that it's the natural notion when the ansatz needs to act on arbitrary inputs (e.g., quantum kernels, quantum data processing).

Practical Universality: What "Enough" Looks Like

The interesting question is not "what is the minimum parameter count for universality" but **"how many parameters give an ansatz expressive enough to solve the problem at hand?"**

For chemistry (VQE on small molecules): - Hartree-Fock state plus a UCCSD ansatz with $O(N^4)$ parameters (for N electrons) is enough to reach chemical accuracy on most small molecules. This is *much* less than full universality.

For optimization (QAOA): - A QAOA ansatz with $O(p)$ parameters at depth p is enough to solve many MaxCut and constraint-satisfaction problems with provable approximation guarantees. The required p is small (often $p = 1$ or 2) for many problems, vastly less than universality.

For machine learning (data-encoded models): - A re-uploading ansatz with $O(L)$ data-encoding layers can express functions of growing complexity. Schuld-Sweke-Meyer (2021) showed that L layers gives Fourier series of degree up to L.

The pattern: **practical ansätze trade universality for trainability**. The right amount of expressivity is just enough to contain the answer, and not more.

The Schuld-Sweke-Meyer Theorem

A particularly clean universality result for quantum machine learning models.

Setup. A quantum model is a function $f(x; \boldsymbol{\theta}) = \langle 0|U^\dagger(x, \boldsymbol{\theta}) M U(x, \boldsymbol{\theta})|0\rangle$, where x is a data input encoded into the circuit, $\boldsymbol{\theta}$ are trainable parameters, and M is an observable.

Theorem (Schuld-Sweke-Meyer 2021). If the data x is encoded via Pauli rotations e^{-ixH}, and the encoding is repeated L times with trainable circuits in between, then $f(x)$ is a **truncated Fourier series** in x of degree $\leq L$. The Fourier coefficients are determined by the trainable circuits; in particular, all Fourier modes up to degree L can be reached by suitable choice of $\boldsymbol{\theta}$.

Implication: with enough re-uploading layers, the model can approximate any continuous function of x on a compact domain (Stone-Weierstrass for trigonometric polynomials). This is a quantum universal approximation theorem in the function-approximation sense.

The catch: the Fourier coefficients are constrained by the structure of the circuit. The model is universal in the sense that *some* coefficients can be chosen freely, but not all combinations are achievable. There are dependencies that the proof doesn't explicitly characterize.

This result is the cleanest connection between quantum circuits and classical function approximation theory, and it's the theoretical foundation for **data re-uploading** as a quantum machine learning architecture.

Universality And Barren Plateaus

The deep tension: **the most universal ansätze are exactly the ansätze most prone to barren plateaus**.

Why: universality (or near-universality) means the ansatz reaches a Haar-like distribution over states. Module 4 showed that Haar-2-design ansätze have exponentially small gradient variance — barren plateau.

Quantitative statement: for an ansatz that is an ϵ-approximate state-2-design,

$$\mathrm{Var}[\partial_j C] \leq \frac{1}{2^n - 1}(\|H\|_2^2 + \epsilon \cdot \mathrm{poly}(n)).$$

As $\epsilon \to 0$ (true 2-design), the bound goes to $1/(2^n - 1)$ — exponentially small.

Conclusion: a fully universal ansatz cannot be trained by gradient descent on a generic Hamiltonian. This is a *theorem*, not a heuristic. The only escape is to use a non-universal (structured, restricted) ansatz.

This is the central theoretical message of Module 8: **universality and trainability are in fundamental opposition**, and any practical VQA strategy must explicitly choose where to land on the spectrum.

The Practical Take

Universal approximation theorems for quantum circuits exist, but they:

1. **Require exponentially many parameters** (state univ: $2 \cdot 2^n - 2$, unitary univ: $4^n - 1$).
2. **Imply barren plateaus** (universal = Haar-like = no gradients).
3. **Are not how successful VQAs actually work** (chemistry uses problem-inspired, optimization uses problem-restricted).

The honest summary: universal approximation is a theoretical curiosity for quantum circuits, not a practical guide. **The practically useful theorems are about restricted ansätze**: how much function approximation can a *specific structured* ansatz do, given its symmetries and depth?

This is the territory of nodes 8.4 (trainability bounds for restricted ansätze) and 8.5 (the explicit tradeoff curve).

Universal Approximation In Classical ML, For Comparison

The classical analogue is much friendlier:

- A neural net with one hidden layer of width $O(\mathrm{poly})$ is a universal approximator (Cybenko, Hornik). The width grows polynomially in the function complexity, not exponentially.
- Backpropagation works regardless of width — no analogue of barren plateaus that scales with width.

- In practice, classical NNs are *vastly* over-parameterized, and that over-parameterization is *helpful*, not harmful (Belkin-Hsu-Ma-Mandal "double descent").

In quantum circuits, **the relationship is inverted**: over-parameterization is harmful, and the right amount of structure is essential. This is one of the fundamental qualitative differences between quantum and classical machine learning, and it shapes everything about how VQAs are designed.

Structural Role

This node is the **theoretical reference point** of Module 8. It establishes what universal approximation means in the quantum setting, explains why universal ansätze are not what we want, and motivates the move to restricted ansätze that is the focus of 8.4 and 8.5.

It also provides the connection to classical machine learning theory (and the cautionary tale about how the classical results do *not* translate directly).

Relations to Other Concepts

- **Cybenko 1989, Hornik 1991** — classical universal approximation.
- **Iten et al. 2016** — explicit universal state preparation circuits.
- **Schuld-Sweke-Meyer 2021** — Fourier series structure of data re-uploading.
- **McClean et al. 2018** — barren plateau theorem (the cost of universality).
- **Module 4 (Barren plateaus)** — the trainability cost.

Worked Example — Counting Parameters For A 4-Qubit Universal Ansatz

For 4 qubits, the projective Hilbert space has real dimension $2 \cdot 2^4 - 2 = 30$.

State-universal ansatz: at least 30 parameters. **Unitary-universal ansatz:** at least $4^4 - 1 = 255$ parameters.

A Hardware-Efficient Ansatz with single-qubit R_X, R_Y, R_Z on each qubit (12 parameters per layer) plus 4 nearest-neighbor CNOTs would need at least $\lceil 30/12 \rceil = 3$ layers for state universality (if the CNOTs effectively connect the parameters). In practice, the SJK expressibility approaches Haar at around 3-5 layers, consistent with this dimension counting.

For unitary universality, you'd need much more: at least 21 layers. In practice, even 21 layers doesn't quite achieve unitary universality for HEA because the circuit's connectivity restricts the reachable unitaries.

The dimension counting is a *lower bound* — actual universality requires the right topology of parameter dependence, which not every counting-correct circuit provides.

Visualization

Pending. A chart showing parameter count vs qubit count for state universality, unitary universality, and "practical" UCCSD/HEA. State universality grows as 2^{n+1}, unitary as 4^n, UCCSD as $O(n^4)$, HEA polynomial in n per layer. Caption: "Universality is exponential. Practical ansätze are polynomial."

Exercises

1. For a 1-qubit ansatz $R_Y(\theta_1)R_X(\theta_2)R_Y(\theta_3)$, is it state-universal? Unitary-universal? Justify.
2. The Schuld-Sweke-Meyer theorem says L data-encoding layers give Fourier degree L. For a function $f(x) = \cos(3x) + 0.5\cos(5x)$, what is the minimum L you need to express it exactly?
3. (VQA) For the LiH molecule (10 qubits in a minimal basis), estimate the parameter count of (a) a state-universal ansatz, (b) a UCCSD ansatz, (c) a Hardware-Efficient Ansatz with 5 layers. Comment on the gap.

Research Extensions

- **Quantitative universality vs trainability tradeoffs** — explicit curves for specific ansatz families.
- **Constrained universality** — universality on a symmetry-restricted subspace (e.g., particle-number sectors).
- **Practical universality with depth annealing** — starting with restricted ansätze and growing toward universality.
- **Universal approximation under noise** — how hardware errors affect the universality results.

Cross-links to Other Courses

- **Approximation Theory** — universal approximation theorems, density results.
- **Functional Analysis** — Stone-Weierstrass theorem, denseness in continuous function spaces.
- **Module 4 (Barren Plateaus)** — the trainability cost of universality.
- **Module 2 (Ansatz Design)** — the practical alternative to universality.

Trainability Bounds

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.4

This is the **trainability counterpart** of nodes 8.1-8.3. Where those nodes asked "how big is the family of states an ansatz can reach?", this one asks "how hard is it to optimize over that family?"

The two questions are not independent. Module 4 already showed that a fully expressive (Haar 2-design) ansatz has barren plateaus. This node makes the relationship **quantitative**: it states the trainability bounds in their precise forms, walks through the most important variants, and connects each one to the corresponding expressivity property.

The result of this node is the inputs to node 8.5, which combines them into the **expressivity-trainability tradeoff curve** that is the central object of Module 8.

What "Trainability" Means

Several closely-related but distinct quantities can serve as "trainability":

1. Gradient variance. $\mathrm{Var}_{\boldsymbol{\theta}}[\partial_j C(\boldsymbol{\theta})]$ for randomly initialized parameters. Small variance = barren plateau = low trainability.

2. Gradient signal-to-noise ratio. $\mathbb{E}[\partial_j C]^2/\mathrm{Var}[\partial_j C]$ — how big is the expected gradient compared to its fluctuation? Low SNR = the gradient is dominated by noise.

3. Expected number of iterations to convergence. Operationally — how many optimizer steps does it take to reach a target accuracy? More iterations = lower trainability.

4. Probability of successful initialization. Out of a random sample of initial parameters, what fraction lead to successful training? Lower fraction = lower trainability.

5. Worst-case landscape geometry. Are there many local minima? Saddle points? Flat regions? More such structures = lower trainability.

The first one — **gradient variance** — is the standard quantitative measure in the VQA literature, and the one most barren plateau theorems are stated for. The others are operationally important but less amenable to formal analysis.

The Master Trainability Inequality

The most general statement, due to Cerezo et al. (2021) and refined by McClean et al. and Holmes et al.:

Theorem (general gradient variance bound). For an ansatz that forms an ϵ-approximate state-2-design, and a global cost $C = \langle\psi|H|\psi\rangle$:

$$\mathrm{Var}_{\boldsymbol{\theta}}[\partial_j C] \leq \frac{2\|H\|_2^2}{2^n - 1} + O(\epsilon).$$

For a true 2-design ($\epsilon = 0$), the bound is exactly $2\|H\|_2^2/(2^n - 1)$ — exponentially small in n.

This is the master inequality. All other trainability bounds in the VQA literature can be viewed as refinements that:

1. **Tighten the bound** for non-2-design ansätze.
2. **Loosen the assumption** to cover specific structured cost functions or ansatz families.

3. **Replace** the 2-design assumption with a Lie-algebraic condition (Ragone et al. 2024).
4. **Add corrections** for noise (Wang et al. 2021).

Bound 1: Original McClean Bound (Global Cost, Deep HEA)

The original barren plateau theorem of McClean et al. 2018:

$$\mathrm{Var}[\partial_j C_{\mathrm{global}}] \in O(2^{-n}).$$

Assumptions: the ansatz at depth $O(n)$ forms an approximate 2-design, the cost is global (sum of single-qubit operators on all qubits), parameters are uniformly initialized.

Implication: the gradient is exponentially small. To resolve it above shot noise requires $O(4^n)$ shots per evaluation. Infeasible.

This is the canonical statement and the one most often cited. Module 4 covered it in detail.

Bound 2: Cerezo Local-Cost Refinement

For a *local* cost function (sum of single-qubit observables), the bound improves dramatically.

Theorem (Cerezo et al. 2021). For $C_{\mathrm{local}} = \sum_i \langle\psi|O_i|\psi\rangle$ where each O_i acts on $O(\mathrm{poly}\log n)$ qubits:

$$\mathrm{Var}[\partial_j C_{\mathrm{local}}] \in \Omega(\mathrm{poly}(1/n)),$$

provided the ansatz is "shallow" (depth $O(\mathrm{poly}\log n)$) and the cost is built from local observables.

Implication: local costs evade the barren plateau in shallow ansätze. The polynomial lower bound on gradient variance is enough for gradient descent to make progress.

Caveat: as the ansatz becomes deeper than $O(\log n)$, even the local-cost bound deteriorates. The refinement only works in the *shallow* regime.

This is the most important practical refinement. It justifies using **depth-restricted, local-cost** VQAs as the default, with global-cost variants only when necessary.

Bound 3: Marrero Entanglement Bound

A more physical view: trainability is related to the **average entanglement** of the ansatz states.

Theorem (Marrero-Kieferová-Wiebe 2021). Let $S(\rho_A)$ be the average entanglement entropy of the reduced state on subsystem A (averaged over parameter initializations). Then

$$\mathrm{Var}[\partial_j C] \leq f(\dim A) \cdot 2^{-S(\rho_A)},$$

where f is a function of subsystem size only.

Implication: high entanglement $\to$ small gradient variance $\to$ barren plateau. The "volume law" entanglement of Haar-random states is exactly the case where S scales linearly in n, giving $2^{-S} = 2^{-O(n)}$ — barren plateau.

This bound provides the *physical mechanism* for the plateau (entanglement spreading) and a recipe for avoiding it: keep the entanglement low (area-law or log-law).

Bound 4: Ragone Lie-Algebraic Bound

The most recent and theoretically clean variant. Replaces the 2-design assumption with a Lie-algebraic condition.

Theorem (Ragone et al. 2024). Let $\mathfrak{g}$ be the dynamical Lie algebra of the ansatz (Section 8.1). Then

$$\mathrm{Var}[\partial_j C] \leq \frac{P_{\mathrm{purity}}(\mathfrak{g}, H)}{\dim \mathfrak{g}},$$

where P_{purity} is a "purity" of the cost Hamiltonian relative to the algebra (a combinatorial quantity that can be computed from $\mathfrak{g}$ and H).

Implication: trainability is governed by the dimension of the dynamical Lie algebra. Small DLA $\rightarrow$ large variance $\rightarrow$ trainable. Large DLA $\rightarrow$ small variance $\rightarrow$ barren plateau.

This is the **cleanest form** of the bound because:

1. It does not assume the ansatz forms a 2-design (which is hard to verify).
2. It works for *any* polynomial-dimension DLA, not just specific families.
3. It connects directly to the expressivity theory (8.1) — the same DLA dimension that controls expressivity also controls trainability.

This unifies expressivity and trainability under a single Lie-algebraic framework, and is the basis for the explicit tradeoff curve in 8.5.

Bound 5: Wang Noise-Induced Plateau

A very different bound, capturing the noise-induced plateau (Module 4 node 4.5).

Theorem (Wang et al. 2021). For an ansatz of depth L under depolarizing noise with rate p per gate:

$$\mathrm{Var}[\partial_j C] \leq C_0 \cdot (1-p)^{2L},$$

for a constant C_0.

Implication: the gradient variance decays *exponentially in depth times noise rate*. Even on a non-2-design ansatz, sufficient depth and noise will create a barren plateau.

Distinction from the unitary plateau: this bound has nothing to do with expressivity or 2-designs. It is a *purely noise-driven* effect. Increasing the ansatz expressivity does not change this bound; only reducing depth or noise does.

This bound matters because it sets the **noise-corrected ceiling** on practical ansatz depth: even if you have a structured (low-DLA) ansatz, deep noise will plateau it.

Bound 6: Local Approximate 2-Designs

A partial-design refinement: ansätze that are approximate 2-designs only on a *subspace*.

Theorem. If the ansatz is a 2-design on a k-dimensional subspace of the full 2^n-dimensional Hilbert space, then for cost functions whose support is contained in that subspace:

$$\mathrm{Var}[\partial_j C] \in \Theta(1/k).$$

Implication: if you can restrict the ansatz to a polynomially-large subspace (e.g., particle-number conservation gives $k = \binom{2n}{n}$), super-polynomial but much less than 2^{2n}), the trainability is only polynomially small, not exponentially.

This is the formal statement of "symmetry-restriction helps." Module 2's UCCSD ansatz uses particle-number conservation; Module 4 explained the heuristic; this bound provides the rigorous version.

The Bounds As A Hierarchy

The bounds form a hierarchy. From most pessimistic to most optimistic:

1. **Wang (noise-induced):** $(1-p)^{2L}$ — applies whenever there is gate noise.
2. **McClean (deep HEA, global cost):** $O(2^{-n})$ — barren plateau.
3. **Ragone (Lie algebraic):** $O(1/\dim \mathfrak{g})$ — depends on ansatz structure.
4. **Marrero (entanglement):** $O(2^{-S})$ — depends on average entanglement.
5. **Subspace 2-design:** $O(1/k)$ — depends on accessible subspace size.
6. **Cerezo (local cost, shallow):** $\Omega(\mathrm{poly}(1/n))$ — trainable.

The right bound depends on the situation:

- **Are you using a deep generic ansatz with global cost?** → McClean / Ragone.
- **Are you using a deep ansatz with local cost?** → Cerezo or Wang (depth/noise).
- **Are you using a structured ansatz with symmetries?** → Subspace 2-design.
- **Are you on noisy hardware?** → Wang dominates.

The practical question is: which of these bounds is *tightest* for your specific case?

How To Use These Bounds In Practice

A workflow for ansatz design using these bounds:

Step 1: Compute (or estimate) the DLA dimension of your candidate ansatz. If polynomial in n, Ragone gives a polynomial bound — likely trainable.

Step 2: Check the cost function locality. If global, you'll need either a shallow ansatz or a structured DLA. If local, Cerezo gives polynomial trainability for shallow circuits.

Step 3: Estimate ansatz entanglement. If average entanglement is volume-law, Marrero predicts a plateau. If area-law or log-law, you're likely fine.

Step 4: Account for noise. Multiply by Wang's $(1-p)^{2L}$ factor. If your depth times noise rate exceeds 1, the noise plateau dominates.

Step 5: Verify with empirical gradient variance. Sample 100 random initializations, compute the gradient norm at each, look at the distribution. If most are below the shot noise floor, you have a plateau in practice.

This workflow is what experienced VQA practitioners do, often informally. The bounds in this node make it formal.

Lower Bounds Vs Upper Bounds

A subtle but important point: **all these are upper bounds on gradient variance**.

An upper bound says "the variance is at most X." If X is small, you're guaranteed a plateau. A lower bound would say "the variance is at least X." If X is large, you're guaranteed *not* to have a plateau.

The literature has many upper bounds (showing plateaus are inevitable in various regimes) and very few lower bounds (showing trainability is guaranteed in various regimes). The Cerezo bound for shallow local-cost circuits is one of the few clean lower bounds.

This asymmetry reflects the difficulty: it's easier to prove "this thing is bad" than "this thing is good." For practical VQA design, the lack of clean lower bounds means you often have to verify trainability empirically rather than rely on theory.

Structural Role

This node is the **trainability theorem dump** of Module 8. It collects the formal trainability bounds, organizes them into a hierarchy, and provides the practitioner workflow for using them.

It directly enables node 8.5 (the explicit expressivity-trainability tradeoff curve), which combines the bounds in this node with the expressivity metrics in 8.1-8.2 to produce the central object of Module 8.

Relations to Other Concepts

- **McClean et al. 2018** — original barren plateau theorem.
- **Cerezo et al. 2021** — local cost refinement.
- **Marrero-Kieferová-Wiebe 2021** — entanglement view.
- **Ragone et al. 2024** — Lie-algebraic theory.
- **Wang et al. 2021** — noise-induced plateaus.
- **Module 4 (Barren Plateaus)** — the qualitative version of all this.

Worked Example — Comparing Bounds On A 6-Qubit Ansatz

Consider a 6-qubit Hardware-Efficient Ansatz with 4 layers and a global cost $C = \langle Z^{\otimes 6} \rangle$.

McClean bound: $\mathrm{Var}[\partial_j C] \leq 2 \cdot 1/(2^6 - 1) = 2/63 \approx 0.032$.

The shot noise floor at $S = 1000$ shots is $\sim 1/\sqrt{S} \approx 0.032$. The gradient is **right at the noise floor**. Training will be very hard.

Cerezo bound (if cost were local): if $C = \sum_i \langle Z_i \rangle$ instead, the bound would be $\Omega(1/n^c)$ for some small c, giving Var $\approx 0.1 - 0.5$ — well above the noise floor. Training would be feasible.

Ragone bound: the DLA of a 4-layer 6-qubit HEA is approximately $2^{12} = 4096$ (full unitary algebra). $P_{\text{purity}} \approx 1$. So Var $\leq 1/4096 \approx 0.0002$. **Tighter than McClean** — predicts a worse plateau than McClean's bound. Empirically, this is consistent with what's observed.

Wang bound at $p = 10^{-3}$ noise: $(1 - 10^{-3})^{2 \cdot 24} \approx 0.95$. Negligible noise contribution at this depth.

Marrero bound: average entanglement of a 4-layer HEA on 6 qubits is approximately the Page value, $S \approx 3$. So $2^{-3} = 0.125$. Looser than McClean but in the same regime.

The verdict: for this configuration, all bounds say "barren plateau." Switching to a local cost (Cerezo) or restricting to a structured ansatz (smaller DLA in Ragone) would make it trainable.

Visualization

Pending. A log-scale plot of gradient variance vs qubit count for: (a) McClean (deep HEA global), (b) Cerezo (shallow HEA local), (c) Ragone (depends on DLA dim), (d) Wang (depends on depth and noise). Each is a different curve, with the shot noise floor marked. Caption: "Different bounds; different regimes; different conclusions."

Exercises

1. For a 4-qubit HEA with 2 layers and global cost $\langle Z_0 Z_1 Z_2 Z_3 \rangle$ with $\|H\| = 1$, what does McClean's bound give? Compare to a shot noise floor at 10000 shots.
2. The Ragone bound is $O(1/\dim \mathfrak{g})$. For a UCCSD ansatz on H_2 (2 electrons in 4 spin orbitals), $\dim \mathfrak{g} = ?$ (Hint: it's the dimension of the particle-number-conserving subalgebra.) What does the bound predict for trainability?
3. (VQA) For a 10-qubit HEA at depth 30 with depolarizing noise $p = 10^{-3}$, is the gradient plateau dominated by McClean (unitary) or Wang (noise)? Show your reasoning.

Research Extensions

- **Lower bounds for structured ansätze** — proving trainability rather than disproving it.
- **Bounds for non-i.i.d. parameter initializations** — most theorems assume uniform; what about smarter init?
- **Bounds under realistic hardware noise** — beyond depolarizing.
- **Bounds for hybrid optimizers** — how does HOPSO change the effective trainability bound?

Cross-links to Other Courses

- **Random Matrix Theory** — Haar measures, t-design moments.
- **Lie Theory** — dynamical Lie algebras (Ragone).
- **Module 4 (Barren Plateaus)** — the qualitative discussion these bounds formalize.
- **Module 6 (Optimization Dynamics)** — how the bounds affect optimizer choice.

The Expressivity-Trainability Tradeoff

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.5

This is the **central node** of Module 8 — the one that combines the expressivity machinery (8.1-8.3) and the trainability bounds (8.4) into a single quantitative statement: **a curve relating how much an ansatz can express to how easily it can be optimized**.

The tradeoff is not a heuristic anymore. It is a *theorem*. The Ragone et al. (2024) Lie-algebraic framework provides the cleanest version, and it says (paraphrasing):

> For an ansatz with dynamical Lie algebra dimension D, the gradient variance is at most $1/D$ and the dimension of the reachable manifold is at most D.

So **expressivity grows linearly with D, trainability shrinks linearly with $1/D$**, and the product (expressivity $\times$ trainability) is bounded.

This is the formal version of every intuition Module 4 and Module 8 have been building toward. This node states it precisely, walks through its implications, and discusses how to navigate the tradeoff in practice.

The Tradeoff In One Equation

For a parameterized circuit with dynamical Lie algebra $\mathfrak{g}$ of dimension D:

$$\underbrace{\dim \mathcal{R}(U) \leq D}_{\text{expressivity bound}} \quad \text{and} \quad \underbrace{\text{Var}[\partial_j C] \leq \frac{P(\mathfrak{g}, H)}{D}}_{\text{trainability bound}}$$

where $P(\mathfrak{g}, H)$ is the cost-purity factor (a quantity that depends on how the cost Hamiltonian aligns with the algebra; typically $O(1)$).

Multiplying:

$$\dim \mathcal{R}(U) \cdot \text{Var}[\partial_j C] \leq P(\mathfrak{g}, H).$$

This is the tradeoff. The product of expressivity (in the dimension sense) and trainability (in the gradient-variance sense) is bounded by a constant. You cannot have both large.

What The Tradeoff Says

A few interpretations:

1. There is no free lunch. Every gain in expressivity comes at a proportional cost in trainability. If you want to reach a larger family of states, you pay with smaller gradients.

2. The boundary is a curve. Plotting (expressivity, trainability) as a 2D point, all achievable ansätze lie below a hyperbola. The hyperbola is the **Pareto frontier** — the boundary where you can't improve one without sacrificing the other.

3. Most ansätze are far from the frontier. The bound is an inequality, not an equality. Many ansätze are strictly worse than the frontier — they have low expressivity *and* low trainability. Only a small subset are *Pareto optimal*.

4. Choosing where to land is the design decision. Given the frontier, the practitioner picks a point. High expressivity, low trainability (deep HEA on small problems where you can afford the optimization cost). Low expressivity, high trainability (UCCSD on chemistry where the symmetries do the work). Or somewhere in the middle.

The Ragone Curve, Explicitly

The Ragone et al. theorem provides an explicit curve.

For an ansatz indexed by a "structure parameter" s (e.g., depth, number of generators, allowed symmetries), as s increases:

- The DLA dimension $D(s)$ increases.
- The reachable manifold dimension grows: $\dim \mathcal{R}(s) \leq D(s)$.
- The gradient variance shrinks: $\mathrm{Var}(s) \leq P/D(s)$.

The achievable region in (expressivity, trainability) space is bounded by:

$$\text{trainability} \cdot \text{expressivity} \leq \text{const.}$$

For specific ansatz families, the curve has been computed. Examples:

Deep HEA family, parameterized by depth L: - $D(L)$ saturates at $4^n - 1$ (full unitary algebra) for $L = O(n)$. - Expressivity grows from polynomial to exponential. - Trainability drops from $O(1)$ to $O(2^{-n})$.

UCCSD family, parameterized by excitation order k: - $D(k)$ grows polynomially in k: $D \sim n^{2k}$. - Expressivity grows polynomially. - Trainability drops polynomially. - *Stays in the polynomial regime* — never hits the exponential plateau.

Symmetry-restricted ansätze, parameterized by the symmetry group G: - D is fixed by the dimension of the G-invariant operator algebra. - Both expressivity and trainability are determined by G, not directly tunable. - The "right" G is problem-specific.

Pareto Frontier And Optimal Ansätze

A Pareto-optimal ansatz is one where you cannot improve expressivity without losing trainability, or vice versa. The Pareto frontier is the curve of such ansätze.

Properties of the frontier:

1. **It is monotone:** moving along the frontier, gaining expressivity always loses trainability and vice versa. No "free improvements."
2. **It is convex** (in the appropriate scaling): the tradeoff has diminishing returns — going from low to medium expressivity costs little trainability, but going from medium to high expressivity costs proportionally more.
3. **Different problems have different optimal points:** the right point on the frontier depends on the cost function (specifically, how much expressivity is needed to represent the optimum).
4. **Most published ansätze are sub-optimal**, but the gap to the frontier is small in the cases studied so far.

Practical takeaway: when designing an ansatz, ask: "is this on the frontier, or below it?" If below, can it be improved (more efficient parameterization) to move it up?

Concrete Tradeoff Examples

The bounds in node 8.4 give numerical predictions you can check against this tradeoff.

Example 1: 6-qubit HEA, depth 1. - Expressivity (DLA dim): ~ 100. - Gradient variance (Ragone): ~ 0.01. - Verdict: low expressivity, moderate trainability. Probably below the frontier.

Example 2: 6-qubit HEA, depth 6. - Expressivity (DLA dim): ~ 4000 (full unitary algebra). - Gradient variance (Ragone): ~ 0.0003. - Verdict: maximum expressivity, minimum trainability. On the (exponential) frontier — barren plateau.

Example 3: 6-qubit UCCSD with 2 electrons. - Expressivity (DLA dim): ~ 35 (single + double excitations subspace). - Gradient variance (Ragone): ~ 0.03. - Verdict: structured, low expressivity, high trainability. On the (polynomial) frontier.

Example 4: 6-qubit Hamiltonian Variational Ansatz, 3 layers. - Expressivity (DLA dim): ~ 200 (problem-specific algebra). - Gradient variance: ~ 0.005. - Verdict: moderate expressivity, moderate trainability. Close to the frontier.

The pattern: **structured (problem-aligned) ansätze are on the frontier; generic ansätze are typically below it**.

Why The Tradeoff Is Tight

A natural question: is the tradeoff *fundamental* (a true theorem about quantum mechanics) or *artifact-of-our-bounds* (only as tight as the bounds, which could be loosened by smarter analysis)?

The honest answer: **partially fundamental, partially artifactual**.

Fundamental part: the relationship between Lie algebra dimension and reachable set is exact (it's a theorem of differential geometry). The relationship between Lie algebra dimension and gradient variance has the right scaling (also a theorem). The product is bounded by a constant that depends only on the cost Hamiltonian's purity.

Artifactual part: the constant $P(\mathfrak{g}, H)$ might be tightenable by smarter analysis. The gradient variance bound is an upper bound — empirically the variance can be smaller, which would make the tradeoff *worse* than predicted (more pessimistic). Lower bounds would be needed to show the tradeoff is tight in both directions.

The 2026 state of the art is that the tradeoff is **definitely valid** as a hyperbolic upper bound, and **probably tight up to constants** for generic ansätze. For special structured ansätze, there may be more room.

Implications For Ansatz Design

The tradeoff turns ansatz design into a constrained optimization problem:

$$\text{maximize trainability}(U) \quad \text{subject to expressivity}(U) \geq E_{\text{required}},$$

where E_{required} is the minimum expressivity needed to contain the answer to the problem at hand.

This formulation makes the design problem clean:

Step 1: Estimate E_{required} from the problem. For chemistry, this is "enough excitations to capture electron correlations." For optimization, "enough expressivity to express the optimal mixing operator." For machine learning, "enough Fourier modes for the data."

Step 2: Find the ansatz family with smallest possible expressivity that still meets E_{required}.

Step 3: Among that family, pick the variant with the largest trainability.

Step 4: Verify empirically.

This is what UCCSD does for chemistry, what QAOA does for optimization, what re-uploading does for machine learning. They are all different *operational embodiments* of the same constrained optimization principle.

Connection To The Thesis

The thesis frames the tradeoff in terms of **regularization**. The argument:

The expressivity-trainability tradeoff says: an unrestricted (highly expressive) ansatz is untrainable. To make it trainable, you must *restrict* it. There are two ways to restrict:

1. Restrict the ansatz — fewer parameters, smaller DLA, more symmetries. This is the *static* restriction, baked into the ansatz design.

2. Restrict the optimizer dynamics — population-based methods, regularized updates, momentum, trust regions. This is the *dynamic* restriction, baked into the optimizer.

The thesis claims **both** are forms of regularization, and the dynamic version is underexploited. Module 5 (regularization) and Module 6 (optimization dynamics) cover the dynamic version; this module covers the static version.

The key insight: **both forms of restriction move you to a different point on the expressivity-trainability tradeoff curve**. Static restriction trades expressivity at design time; dynamic restriction trades effective expressivity at optimization time. They are complementary.

What The Tradeoff Doesn't Say

A few things the tradeoff curve does *not* tell you:

1. It doesn't tell you where on the curve you should be. That depends on the problem.

2. It doesn't tell you which structured ansatz is best. Many ansätze can be at the same point on the curve, but with different fitness for different problems.

3. It doesn't tell you about local minima. The tradeoff is about *global* trainability properties (gradient variance). It doesn't address whether the landscape has many local minima.

4. It doesn't tell you about noise. Hardware noise modifies the effective tradeoff (Wang's bound) but the noise correction is on top of the unitary tradeoff.

5. It doesn't predict generalization. For VQA-as-machine-learning, there's a separate question about how well the ansatz generalizes from training data, which the tradeoff doesn't address.

These are all separate questions that nodes 8.6 (data encoding) and 8.7 (quantum advantage) take up.

Structural Role

This node is the **central theorem** of Module 8. It combines the expressivity machinery (8.1-8.3) with the trainability bounds (8.4) to produce the explicit tradeoff curve, and it provides the practical workflow for designing ansätze in light of the tradeoff.

It is the conceptual climax of Module 8, and the main reference point for Module 9 (Hardware Noise) and Module 10 (Adaptive Control), which discuss how the tradeoff plays out in real deployment.

Relations to Other Concepts

- **Ragone et al. (2024)** — the primary reference for the explicit tradeoff.
- **Holmes et al. (2022)** — earlier connection between expressibility and trainability.
- **Sim-Johnson-Killoran (2019)** — the empirical version of the same intuition.
- **No-free-lunch theorems (Wolpert-Macready 1997)** — classical analogue from optimization.
- **Module 4 (Barren plateaus)** — the qualitative version.

Worked Example — The Tradeoff For A Family Of Ansätze

Consider 6-qubit ansätze with single and double Pauli rotations, parameterized by the number of layers L.

L	DLA dimension	Expressivity (manifold dim)	Variance bound (P/D)	Where on curve
1	24	24	0.04	Below frontier
2	100	100	0.01	Approaching frontier
3	400	400	0.0025	On frontier
4	1500	~1500	0.0007	On frontier
5	3500	3500	0.0003	Saturated
6+	4096	4096 (max)	0.00024	Plateau regime

The product (expressivity × variance) hovers around 1 for layers 3-6, confirming the tradeoff. Below $L = 3$, the product is below 1 — the ansatz is sub-optimally compressing.

To use this ansatz for a problem requiring expressivity ~400 (manifold dimension), $L = 3$ is the right choice. Going to $L = 5$ or $L = 6$ gains nothing (the ansatz can already represent the answer) but loses an order of magnitude in trainability.

This is the kind of analysis the tradeoff curve enables.

Visualization

Pending. A 2D plot of trainability (y-axis, log scale) vs expressivity (x-axis, log scale) with the Pareto frontier curve, several ansatz points (UCCSD, HEA-1, HEA-3, HEA-deep, problem-aligned), and the "below frontier" region shaded. Caption: "The tradeoff is real. Most ansätze are on the curve. The right point depends on your problem."

Exercises

1. For an ansatz with DLA dimension 50 and cost-purity 1, what is the maximum gradient variance? Is this trainable at 1000 shots?
2. The tradeoff says (expr × var) ≤ const. For two different cost Hamiltonians (one global, one local), the constants differ. Which has the larger constant — global or local? Explain.
3. (VQA) Design a 4-qubit ansatz with target expressivity (manifold dimension) ~20 and predict its gradient variance from the tradeoff. Verify by computing the DLA dimension explicitly.

Research Extensions

- **Tighter constants in the tradeoff** — pinning down $P(\mathfrak{g}, H)$ for various ansatz-Hamiltonian combinations.
- **Tradeoff under noise** — combining Ragone with Wang to get the effective tradeoff under hardware noise.
- **Tradeoff for non-standard cost functions** — non-Hermitian observables, multi-objective costs.
- **Tradeoff and Pareto-optimal ansatz search** — automated discovery of ansätze on the frontier.

Cross-links to Other Courses

- **Multi-objective optimization** — Pareto frontiers in classical optimization.
- **Information theory** — capacity-rate tradeoffs (Shannon's noisy channel theorem is the classical analogue).
- **Module 4 (Barren plateaus)** — the qualitative root of the tradeoff.
- **Module 5 (Regularization)** — the dynamics-side complement.

Data Encoding and Expressivity

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.6

So far, Module 8 has treated expressivity in terms of *parameter* dependence: how many states the ansatz can reach as $\boldsymbol{\theta}$ varies. This node generalizes to the **machine-learning setting**, where the ansatz also depends on a *data input* x.

In quantum machine learning (QML), the circuit becomes $U(x, \boldsymbol{\theta})$ — a function of both data and trainable parameters. The expressivity questions are now:

1. **What functions $f(x)$ can the model express** as x varies?
2. **How does the choice of data encoding affect the function class?**
3. **Is there a tradeoff between data-encoding expressivity and trainability** analogous to the parameter-side tradeoff?

The answers reveal that data encoding is *the* key design choice in QML — more important than the parameterized part. The encoding determines the function class entirely; the trainable part only selects within it.

This node walks through the major encoding strategies, the Schuld-Sweke-Meyer Fourier characterization, and the implications for QML model design.

Why Data Encoding Matters

In classical ML, data is fed to a neural network via a single input layer. The network's expressivity grows with depth and width regardless of how the input was fed in (modulo choice of features).

In quantum ML, **the encoding is part of the model**. Different encoding choices give *fundamentally different* function classes, even with identical trainable layers. There is no "universal feature encoding" that lets the trainable layers do all the work.

The reason: a quantum state $|\psi(x)\rangle$ is a *nonlinear* function of x (because of the cosines and sines in the rotation gates). The set of functions you can express as expectation values $\langle\psi(x)|M|\psi(x)\rangle$ is constrained by what kinds of nonlinearities the encoding produces. Trainable parameters can mix and combine these nonlinearities, but they can't introduce nonlinearities that aren't already in the encoding.

So **encoding choice = function class**. Get it wrong and no amount of training will help.

The Major Encoding Strategies

The literature has converged on a few standard strategies.

1. Basis encoding

$$x \in \{0,1\}^n \to |x\rangle.$$

Each bit of x becomes a computational basis state. Simple but only works for binary inputs.

Pros: trivial to implement; preserves all information. **Cons:** doesn't generalize to continuous inputs; doesn't introduce useful nonlinearities.

2. Angle encoding

$$x \in \mathbb{R}^n \to \bigotimes_{i=1}^{n} R_Y(x_i)|0\rangle.$$

Each component of x becomes a rotation angle on a separate qubit. The state is a product of single-qubit rotations.

Pros: simple, continuous-friendly, hardware-friendly. **Cons:** the function class is restricted to first-order Fourier modes (linear in $\cos x_i, \sin x_i$).

3. Amplitude encoding

$$x \in \mathbb{R}^{2^n}, \|x\| = 1 \to \sum_i x_i|i\rangle.$$

The data vector becomes the amplitudes of a quantum state. **Exponential compression** — 2^n dimensional vectors fit in n qubits.

Pros: maximally compact, supports arbitrary inputs. **Cons:** state preparation is expensive — $O(2^n)$ gates in general. Often a bottleneck.

4. Re-uploading encoding

$$x \in \mathbb{R}^d \to U_L(\boldsymbol{\theta}_L)S(x)U_{L-1}(\boldsymbol{\theta}_{L-1})S(x)\cdots S(x)U_0(\boldsymbol{\theta}_0)|0\rangle,$$

where $S(x)$ is a data-encoding layer (e.g., a tensor product of $R_Y(x_i)$) repeated L times, interleaved with trainable layers $U_l(\boldsymbol{\theta}_l)$.

Pros: the function class grows polynomially with L — Fourier modes up to degree L become accessible. Universal function approximator in the limit $L \to \infty$. **Cons:** depth scales with desired Fourier degree.

5. Hamiltonian encoding

$$x \in \mathbb{R} \to e^{-ixH}|0\rangle,$$

where H is a problem-specific Hamiltonian. The data is encoded as evolution time.

Pros: physically natural, exploits quantum dynamics directly. **Cons:** Hamiltonian needs to be implementable; expressivity depends on H's spectrum.

The dominant choice in 2026 QML is **re-uploading**, because it gives the most controllable expressivity. Angle encoding is the simplest baseline.

The Schuld-Sweke-Meyer Theorem (Detailed)

The most important theoretical result about data encoding is the **Fourier characterization** of re-uploading models.

Setup. Consider a 1D input $x \in \mathbb{R}$ and a re-uploading model

$$f(x;\boldsymbol{\theta}) = \langle 0|U^\dagger(x,\boldsymbol{\theta})MU(x,\boldsymbol{\theta})|0\rangle,$$

where the data is encoded via e^{-ixH} gates with Hamiltonian H, and there are L such encoding gates interleaved with trainable layers.

Theorem (Schuld-Sweke-Meyer 2021). $f(x;\boldsymbol{\theta})$ is a *truncated Fourier series* in x:

$$f(x;\boldsymbol{\theta}) = \sum_{\omega\in\Omega} c_\omega(\boldsymbol{\theta}) e^{i\omega x},$$

where the **frequency set** Ω is determined by the spectrum of H and the depth L, and the **Fourier coefficients** $c_\omega(\boldsymbol{\theta})$ are determined by the trainable circuits.

Frequency set details: if H has eigenvalues $\{\lambda_i\}$, then Ω is the set of integer linear combinations of differences $\lambda_i - \lambda_j$ with at most L terms.

Implications:

1. **The encoding determines what frequencies are reachable.** Different H gives different Ω.
2. **The depth controls the Fourier degree.** L layers give frequencies up to roughly degree L.
3. **The trainable parameters control coefficients, not frequencies.** No amount of training will give you a frequency that isn't in Ω.
4. **The function class is fully characterized.** Two re-uploading models with the same encoding and depth have the same function class — only the coefficient flexibility may differ.

This is the cleanest theoretical result in QML expressivity, and it reframes data encoding as **frequency-set design**: pick the encoding that gives you the frequencies you need.

Multi-Dimensional Inputs

For $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, the Fourier characterization generalizes:

$$f(\mathbf{x};\boldsymbol{\theta}) = \sum_{\boldsymbol{\omega}\in\Omega^d} c_{\boldsymbol{\omega}}(\boldsymbol{\theta}) e^{i\boldsymbol{\omega}\cdot\mathbf{x}},$$

where Ω^d is now a *set of frequency vectors* rather than scalars.

The structure of Ω^d depends on:

1. Whether the data components are encoded in **the same qubit** (in which case the frequencies couple) or **different qubits** (in which case the frequencies separate).
2. The **depth** of the re-uploading.
3. The **mixing** of data components in the trainable layers (a non-trivial entanglement layer between encoding gates can mix frequencies that would otherwise be separable).

This multi-dimensional case is more complex and less fully characterized than the 1D case, but the qualitative picture is the same: **encoding determines what's expressible; trainable parameters fine-tune within that space**.

The Encoding-Trainability Tradeoff

There is an analogue of the parameter-side tradeoff (8.5) for the data-encoding side.

Statement (informal). Increasing the data-encoding depth gives a larger Fourier set (more expressivity) but, all else equal, also brings the model closer to the barren plateau regime in the parameters. So there is a similar tradeoff: more data-encoding depth = more function expressivity = harder to train.

Quantitative version. Schuld and Killoran (2022) showed that for re-uploading models on n qubits with L encoding layers, the gradient variance scales as $O(2^{-nL})$ for generic encoding and trainable layers. The plateau is *both* in qubit count and encoding depth.

Mitigation: as in the parameter-side tradeoff, structured encodings (with restricted frequency sets) have better trainability. The right encoding for QML is the one that includes the frequencies needed for the data, but no more.

Encoding As Feature Engineering

A useful classical analogy: **data encoding in quantum ML is like feature engineering in classical ML**.

In classical ML, a kernel or feature map $\phi(x)$ lifts the data into a feature space where a linear (or simple) classifier can separate it. The expressive power of the model is largely determined by the feature map, not the classifier.

In quantum ML, the data encoding plays the same role: it lifts x into a quantum state $|\psi(x)\rangle$, where the trainable observable $\langle M(\boldsymbol{\theta})\rangle$ performs the "linear classification."

The implications:

1. **Expressivity comes mostly from the encoding.** If the encoding is linear in x, the model is at most a linear function of trigonometric features.
2. **Universal QML requires universal encoding.** If you want to learn any function, you need a re-uploading encoding with enough Fourier modes.
3. **Domain knowledge goes into the encoding.** Just as in classical ML, choosing the right features (encoding) for the problem is the most important design decision.

This analogy is not perfect — quantum encodings introduce specifically *trigonometric* nonlinearities that don't have a clean classical counterpart — but it's close enough to import most of the intuition from classical feature engineering.

The Quantum Kernel Perspective

A related view: the encoding defines a **quantum kernel**.

$$K(x, x') = |\langle\psi(x)|\psi(x')\rangle|^2.$$

This is an inner product between encoded data points, computable on a quantum computer. A kernel-based classifier (SVM, ridge regression) can be applied directly.

Properties of quantum kernels:

1. **They capture the encoding's similarity structure.** Two inputs that map to similar quantum states have high kernel value.
2. **They can be exponentially powerful.** For some encodings, the quantum kernel computes inner products in an exponentially-large feature space (related to the IQP encoding of Havlicek et al. 2019).
3. **They sidestep the trainability problem.** Kernel methods don't require gradient descent over a deep parameterized circuit — you compute the kernel matrix and run a classical SVM.

The downside: kernel methods scale as $O(N^2)$ with dataset size N, where N is the number of training examples. For large datasets, this is prohibitive.

The kernel view is the clean theoretical alternative to variational QML, and recent work (Schuld 2021, Liu et al. 2021) suggests it's often *better* than variational training when the right kernel is available.

Practical Encoding Recommendations

For practitioners building QML models in 2026:

For low-dimensional regression / classification ($d < 10$): - Re-uploading with angle encoding, $L = 2 - 5$ layers. - Frequency set design: pick H so the frequencies match the expected smoothness of the target function.

For high-dimensional data ($d > 100$): - Amplitude encoding (if state preparation is feasible). - Or aggressive feature reduction (PCA) before encoding.

For structured data (images, time series): - Hamiltonian encoding with a structured H (e.g., a translation-invariant one for images). - Or specialized encodings (data convolutional layers, quantum CNNs).

For exploratory work: - Start with simple angle encoding to establish a baseline. - Add re-uploading layers until performance plateaus or trainability degrades. - Switch to kernel methods if variational training stalls.

The dominant message: **think about encoding before trainable parameters**. The encoding sets the ceiling on what you can learn; the parameters only navigate within that ceiling.

Structural Role

This node is the **machine-learning extension** of Module 8. It generalizes the expressivity framework from parameters to data inputs, introduces the Schuld-Sweke-Meyer Fourier characterization, and establishes encoding choice as the primary design decision in QML.

It enables node 8.7 (Quantum Advantage Criteria) by clarifying what kinds of functions quantum models can express that classical models can't — the prerequisite for any advantage claim.

Relations to Other Concepts

- **Schuld-Sweke-Meyer (2021)** — the Fourier characterization theorem.
- **Havlicek et al. (2019)** — IQP-based quantum kernels.
- **Schuld (2021)** — quantum kernel methods.
- **Pérez-Salinas et al. (2020)** — original re-uploading paper.
- **Module 4 (Barren plateaus)** — the trainability cost of expressive encodings.

Worked Example — Frequency Set For A Simple Re-Uploading Model

Consider a 1-qubit re-uploading model:

$$U(x, \boldsymbol{\theta}) = R_Y(\theta_4) e^{-ixZ} R_Y(\theta_3) e^{-ixZ} R_Y(\theta_2) e^{-ixZ} R_Y(\theta_1).$$

Three data encoding gates with $H = Z$, so the eigenvalues are $\{+1, -1\}$ and the differences are $\{2, 0, -2\}$.

Compute the frequency set. With $L = 3$ encoding layers and frequency differences $\{0, \pm 2\}$, the reachable frequencies are integer combinations of $\{2, 0, -2\}$ with at most 3 terms. This gives $\{-6, -4, -2, 0, 2, 4, 6\}$ — degrees up to 6.

Function class. $f(x; \boldsymbol{\theta})$ is a Fourier series with frequencies in $\{-6, -4, -2, 0, 2, 4, 6\}$ — i.e., a degree-6 trigonometric polynomial in $2x$.

Function expressibility. Can this model express $\sin(3x)$? The frequencies in $\sin(3x)$ are ± 3, which are *not* in the model's frequency set (which only has even frequencies). So no — the model cannot exactly express $\sin(3x)$.

To get $\sin(3x)$, you'd need $H = 3Z/2$ so the frequency differences include ± 3, or a different encoding that produces odd frequencies.

This is the kind of analysis you do *before* training: check that your encoding can represent the function family you care about.

Visualization

Pending. A diagram showing two re-uploading models: (a) with $L = 1$ encoding layer, frequency set $\{0, \pm 2\}$, expressing only low-frequency trigonometric functions; (b) with $L = 4$ encoding layers, frequency set $\{0, \pm 2, \pm 4, \pm 6, \pm 8\}$, expressing higher-frequency functions. Side by side with target function plots that the models can/can't reach. Caption: "Encoding sets the frequency budget. Training spends it."

Exercises

1. For a 1D re-uploading model with $H = X$ and $L = 2$ encoding layers, compute the frequency set. Which simple functions $\sin(\omega x), \cos(\omega x)$ can it represent for integer ω?
2. Can amplitude encoding represent the function $f(x) = x^2$? If yes, how? If no, what would you change?
3. (VQA) For a 2D classification task with circular decision boundary, design a re-uploading encoding that has the right frequency content. Which value of L is sufficient?

Research Extensions

- **Frequency-set design** — algorithmic methods for picking H and L to match a target function class.
- **Encoding-trainability optimization** — joint optimization of encoding and parameters.
- **Quantum kernels with provable advantage** — encodings that produce kernels not classically computable.
- **Multi-dimensional Fourier characterization** — extending Schuld-Sweke-Meyer to high-dimensional inputs.

Cross-links to Other Courses

- **Fourier Analysis** — trigonometric polynomials, Fourier series, Stone-Weierstrass.
- **Kernel Methods (classical ML)** — kernel SVMs, RKHS, feature maps.
- **Module 4 (Barren plateaus)** — trainability cost of deep encodings.
- **Module 8 Section 8.7 (Quantum Advantage)** — what encodings enable provable quantum advantage.

Quantum Advantage Criteria

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.7

This is the **closing node** of Module 8 — and arguably the most important one for anyone wondering whether all of this VQA machinery actually *matters*.

The question is simple: **when does a VQA actually beat a classical algorithm at the same task?**

The answer, as of 2026, is honest and uncomfortable: **almost never, with current hardware**. There are theoretical regimes where quantum advantage is provable. There are heuristic regimes where it's plausible. There are application-specific regimes where it's claimed. But for the vast majority of VQA experiments published in the past decade, the honest assessment is that a classical algorithm of comparable effort would do at least as well.

This node walks through the criteria for quantum advantage, the regimes where it's been claimed, the regimes where it's been *disproved* (a sobering category), and the open questions about where future advantage might come from. It is the essential reality check for the rest of Module 8 and indeed for the entire VQA literature.

Three Notions Of Quantum Advantage

The phrase "quantum advantage" means different things in different contexts. The three main notions:

1. Asymptotic advantage (provable). A quantum algorithm has an asymptotically better runtime (poly vs super-poly, or poly vs exp) than the best classical algorithm for the same task. Examples: Shor's algorithm, Grover's algorithm. Holds for all sufficiently large inputs.

2. Empirical advantage (heuristic). On a specific benchmark or instance, the quantum algorithm produces better results (lower cost, faster) than a specific classical baseline. Holds only for the specific benchmark; may not generalize.

3. Sample complexity advantage (practical). The quantum algorithm uses fewer samples (less data) to achieve a given accuracy than any classical algorithm. Especially relevant for QML, where data is expensive.

VQAs are usually assessed in the second sense (empirical) or, more recently, the third (sample). The first (asymptotic) is rarely on offer for VQAs, because the variational structure makes asymptotic analysis hard.

What Provable Asymptotic Advantage Requires

For a VQA to give provable asymptotic advantage, several conditions must hold:

1. The classical baseline is well-characterized. You must know the best classical algorithm for the same task, or have a complexity-theoretic lower bound. For most problems VQAs target (chemistry ground states, optimization), the best classical algorithms are heuristics — there's no clean lower bound.

2. The quantum algorithm has a provable runtime. VQAs have variational structure with classical optimization in the loop, and the convergence of the classical optimizer is itself unprovable in general. So the *whole* VQA runtime is unprovable.

3. The quantum part actually does something exponentially hard for classical methods. Many quantum-inspired classical algorithms (DMRG, neural network states, Monte Carlo) can do what shallow VQAs do, sometimes faster.

Net result: no VQA has provable asymptotic advantage as of 2026. Several have *plausible* asymptotic advantage based on conjectured complexity classes (e.g., assuming $P \neq NP$), but no rigorous proof.

Empirical Advantage And The Dequantization Problem

The empirical advantage story is complicated by **dequantization** — the discovery that many quantum algorithms thought to give exponential speedup actually have classical algorithms with similar performance.

Notable dequantizations:

- **Tang's quantum-inspired classical recommendation algorithm (2018):** showed that the quantum recommendation algorithm of Kerenidis and Prakash, which appeared to give exponential speedup, has a classical analogue with polynomial-time runtime.
- **Quantum-inspired classical PCA, regression, etc.** Many "quantum machine learning" speedups have classical analogues.
- **Tensor network simulations** of VQAs that claimed empirical advantage have repeatedly shown that DMRG or MPS methods reach the same accuracy with less compute.

The pattern: every time a VQA claims empirical advantage, the classical ML/numerical community responds with a quantum-inspired algorithm that closes the gap.

This is sobering. It does *not* mean quantum advantage is impossible — it means **the bar is high**. To claim a quantum advantage, you need to show your problem instance is hard for *all* known classical methods (DMRG, NN-states, Monte Carlo, etc.), not just one specific baseline.

When Quantum Advantage Is Plausible For VQAs

Despite the dequantization caution, there are regimes where VQA quantum advantage remains plausible:

1. Strongly correlated electron systems beyond DMRG range. DMRG works well for 1D and quasi-1D systems but degrades in 2D and 3D. For 2D Hubbard model at intermediate doping, no classical method scales well. A VQE on a 2D Hubbard ansatz may be the right tool — *if* the VQA can be made to converge.

2. Quantum chemistry with strong static correlation. Coupled-cluster methods (CCSD, CCSD(T)) fail for systems with strong static correlation (transition metals, bond-breaking). UCCSD via VQE might handle these — but the trainability and noise problems make it hard to demonstrate.

3. Sampling problems related to BosonSampling/IQP. Some sampling tasks are *provably* classically hard (Aaronson-Arkhipov for BosonSampling, similar for IQP circuits). These are not VQAs in the strict sense, but they share architectural features. A VQA whose output distribution is in this hardness class would inherit the advantage.

4. Quantum simulation of dynamics. Time evolution under a Hamiltonian is a natural quantum task. If the VQA prepares states involved in a hard simulation problem, the broader simulation pipeline can give advantage. The VQA part is typically state preparation, not the simulation itself.

5. Tasks with quantum-data inputs. If the data x is itself quantum (e.g., outputs of a quantum sensor), classical methods need to first measure and reconstruct, which is exponentially expensive. A quantum ML model can process quantum data directly, with provable sample complexity advantage (Huang et al. 2022).

6. Specific structured problems with hidden symmetries. Some VQAs exploit problem-specific structure (e.g., translation invariance, particle conservation) in ways classical algorithms don't naturally exploit. For these problems, a custom-designed VQA may beat generic classical methods.

These six are the **realistic** regimes for VQA advantage. None has been definitively demonstrated as of 2026, but all are active research areas.

The Sample Complexity Frontier

A more recent and more promising frontier: **sample complexity advantages**.

Setup. Suppose you want to learn some property of an unknown quantum state or process. A classical algorithm needs many samples (measurements) to reconstruct the state and then compute the property. A quantum algorithm can sometimes compute the property directly, with exponentially fewer samples.

Results (Huang-Kueng-Preskill 2020, Aharonov et al. 2022):

- **Shadow tomography:** $O(\log d)$ samples suffice for many properties of a d-dimensional state, vs $O(d)$ classically.
- **Quantum learning advantage:** for certain learning tasks (e.g., learning quantum process tomography), a quantum learner uses exponentially fewer samples than any classical learner, and this is *provable*.
- **Quantum data analysis:** the same task can require exponentially less data with quantum access.

Connection to VQAs: if a VQA is part of a sample-efficient protocol (e.g., learning a property of a state that the VQA prepared), the sample complexity advantage carries over. This is one of the cleanest theoretical justifications for VQA-based ML in 2026.

Caveat: sample complexity is not the same as runtime complexity. A protocol with low sample complexity may still have high computational complexity. The advantage is real but limited.

What Disproves Quantum Advantage

A useful counter-frame: what kinds of evidence *would* disprove a VQA quantum advantage claim?

1. **A classical algorithm matches the quantum algorithm's performance.** (Dequantization.)
2. **The classical baseline was incorrectly chosen.** Many "advantage" claims compare against weak baselines.
3. **The quantum algorithm doesn't actually scale.** Many VQAs are demonstrated at small n and don't extrapolate.
4. **The quantum algorithm is dominated by classical processing.** Some VQAs do most of their work classically (in the optimizer) and the quantum part is incidental.
5. **The benchmark doesn't reflect a real problem.** Synthetic benchmarks designed to be quantum-friendly don't transfer to real applications.

A *credible* quantum advantage claim must address all five of these objections. Most published VQA-advantage claims address one or two and ignore the rest.

This is the standard for the field in 2026, and it has gotten significantly stricter over the past five years as dequantization has matured.

The Honest 2026 Assessment

Putting it all together, here is the honest 2026 picture for VQA quantum advantage:

1. No proven asymptotic advantage for any VQA on any natural problem.

2. Several plausible empirical advantages in narrow regimes: - Strongly correlated chemistry / condensed matter beyond classical reach. - Sample complexity advantages for quantum data. - Specific sampling problems inherited from quantum supremacy circuits.

3. Many *disproven* advantages in broad regimes: - Most QML claims for classical data have been dequantized or matched by classical baselines. - Most VQA chemistry below 30 qubits has classical analogues that perform as well or better. - Most QAOA claims for combinatorial optimization don't hold up against modern classical heuristics.

4. The hardware bottleneck. Even if a VQA has theoretical advantage, current hardware (NISQ, ~100 qubits, ~1% noise per gate) cannot demonstrate it. The advantage *might* appear at fault-tolerant scales, but that is years away.

5. The most active research frontier: finding provable sample complexity advantages for VQAs on quantum data, where classical methods are exponentially worse by structural necessity.

This is uncomfortable but it is the field's current state. Anyone working on VQAs should know it.

What Would Change The Picture

For VQA quantum advantage to become *demonstrated* (not just plausible), several things would need to happen:

1. Better hardware. Lower noise ($p < 10^{-4}$ per gate), more qubits ($n > 200$ high-fidelity), and longer coherence times. This is the most important factor and the hardest to control.

2. Better ansätze. Ansätze designed using the expressivity-trainability tradeoff (Module 8.5), with explicit symmetry restrictions matched to the problem. This is a research area that this thesis contributes to.

3. Better optimizers. Hybrid optimizers (HOPSO, Module 6.8) and dynamics-side regularization (Module 5) that can navigate the difficult landscapes. Also a thesis contribution area.

4. Honest classical baselines. The community needs to stop comparing VQAs against weak classical baselines (random search, untuned SGD) and start comparing against state-of-the-art (DMRG, neural quantum states, professional optimization solvers).

5. Focus on quantum-data tasks. Where the input is naturally quantum, the comparison is more honest because classical methods lose information at the input boundary.

If all five happen, VQAs may demonstrate genuine advantage in the late 2020s or 2030s. If only some happen, VQA advantage will remain a theoretical possibility for longer.

Closing Module 8

Module 8 has been a long arc:

- **8.1-8.3:** what expressivity means (definitions, metrics, universal approximation).
- **8.4:** what trainability means (gradient variance bounds, hierarchy of theorems).
- **8.5:** the explicit tradeoff curve relating them (Ragone Lie-algebraic theory).
- **8.6:** the data-encoding extension for quantum machine learning.
- **8.7 (this node):** the honest reality check on whether any of this gives quantum advantage.

The narrative thread: **the expressivity-trainability tradeoff is the central design constraint for VQAs**, and within that constraint, only narrow regimes plausibly give quantum advantage. The thesis's contribution is to provide tools (regularization, hybrid optimizers, dynamics-side framing) for navigating the constraint better — to find more of those narrow regimes, and to make them work in practice.

Module 9 (Hardware Noise Models) and Module 10 (Adaptive Control Systems) take this theoretical foundation and ask what it looks like in real hardware deployment.

Structural Role

This node is the **reality check** of Module 8 and of the VQA section of the course as a whole. It states the criteria for quantum advantage, walks through what's been claimed and disproven, and gives an honest 2026

assessment of the field's state.

It is the natural endpoint of the theoretical arc: having built up the machinery for understanding VQAs in detail, it asks the meta-question of whether VQAs are *worth* understanding in detail, and answers "yes, but with caveats."

Relations to Other Concepts

- **Tang (2018)** — quantum-inspired classical recommendation algorithm, the start of dequantization.
- **Aaronson-Arkhipov BosonSampling** — sampling-based quantum supremacy.
- **Huang-Kueng-Preskill (2020)** — classical shadows and learning quantum data.
- **Aharonov et al. (2022)** — provable sample complexity advantages.
- **Module 4 (Barren Plateaus)** — the trainability obstacle that constrains advantage.

Worked Example — Comparing VQE Against DMRG On A Spin Chain

Consider a 20-site Heisenberg spin chain — a benchmark VQA target.

Classical baseline: DMRG. - Finds the ground state to chemical accuracy in seconds on a laptop. - Scales to ~100 sites without difficulty. - Has been the gold standard for 1D spin systems for 30 years.

VQE attempt (2026 hardware, ~100 qubits, 1% noise): - Hardware-Efficient Ansatz, depth ~10. - 1000-shot gradient estimates, Adam optimizer. - Converges (after careful initialization) to ~1% accuracy. - Total runtime: hours of quantum compute time.

Comparison: - DMRG: seconds, much higher accuracy. - VQE: hours, lower accuracy. - DMRG wins decisively.

For 2D Heisenberg (20×20 lattice): - DMRG accuracy degrades sharply with width — limited to ~10 wide. - VQE on 400 qubits: not yet achievable on real hardware. - *In principle*, this is where VQE could win, if hardware were available.

The honest assessment: for benchmarks where we *can* compare, classical methods win. For benchmarks where we *can't* compare yet (very large 2D lattices), nobody knows. Hardware progress will determine the outcome.

Visualization

Pending. A 2D plot with axes "system size" and "accuracy" showing the regimes where DMRG, neural network states, Monte Carlo, and VQE are competitive. DMRG dominates 1D, NN-states dominate moderate dimensions, Monte Carlo dominates fermionic systems, and a small region in the upper-right is "where VQE might eventually win." Caption: "Quantum advantage is a small slice of problem space, currently empty."

Exercises

1. List three problems where, in principle, a VQA could beat the best known classical algorithm. For each, identify the classical baseline you're comparing against.
2. The dequantization argument says that many quantum ML speedups have classical analogues. Can amplitude encoding be dequantized? Explain.
3. (VQA) For a 30-qubit VQE on a chemistry molecule (e.g., LiH), estimate the total quantum compute time needed to reach chemical accuracy at current noise levels. Compare to DMRG's runtime on the same problem.

Research Extensions

- **Provable quantum advantage on quantum data** — extending the Huang et al. results to broader classes of problems.
- **VQA + classical preprocessing pipelines** — using VQAs as one component of a larger workflow, where the advantage may be in the integration.
- **Honest benchmarking standards** — methodology for VQA comparisons that resists motivated reasoning.
- **Quantum-classical hybrid advantage** — regimes where the joint quantum + classical algorithm beats either alone.

Cross-links to Other Courses

- **Computational Complexity** — BQP, NP, polynomial hierarchy, complexity-class separations.
- **Classical Numerical Methods** — DMRG, neural quantum states, Monte Carlo (the competitors).
- **Module 9 (Hardware Noise Models)** — the hardware limits on advantage.
- **Module 4 (Barren Plateaus)** — the trainability obstacle that constrains advantage.

Module 9 — Hardware Noise Models

Quantum Noise Channels

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.1

This is the **opening node** of Module 9. Modules 1-8 have mostly treated VQAs as if they ran on ideal quantum computers: unitary evolution, exact measurements, no decoherence. That picture is useful for theory but wrong in every practical sense. **Real VQAs run on noisy hardware**, and noise changes the analysis of essentially every module:

- It shifts the cost landscape (Modules 5, 6).
- It creates new barren plateaus (Module 4.5).
- It modifies the expressivity-trainability tradeoff (Module 8.5).
- It introduces drift that adaptive controllers must reject (Module 7.6, Module 10).

Module 9 is the module that takes noise seriously. It builds up the noise framework from first principles — quantum channels, specific noise models, gate error budgets, mitigation techniques — and by the end of the module, you can read any experimental VQA paper and understand what noise assumptions are in play.

This node introduces the mathematical framework: **quantum channels**, the operator-sum (Kraus) representation, and the Stinespring dilation. Noise in VQAs is modeled as a quantum channel applied after each gate (or after the whole circuit), and everything downstream builds on this foundation.

What A Quantum Channel Is

A **quantum channel** $\mathcal{E}$ is a linear map from density matrices to density matrices:

$$\mathcal{E} : \rho \mapsto \mathcal{E}(\rho),$$

satisfying three properties:

1. **Linear:** $\mathcal{E}(a\rho_1 + b\rho_2) = a\mathcal{E}(\rho_1) + b\mathcal{E}(\rho_2)$.
2. **Trace-preserving:** $\mathrm{Tr}(\mathcal{E}(\rho)) = \mathrm{Tr}(\rho)$ for all ρ.
3. **Completely positive:** $(\mathcal{E} \otimes I_n)(\rho)$ is positive-semidefinite for all ρ and all $n \geq 0$.

Together these are called **CPTP maps** (completely positive trace-preserving). A quantum channel is the most general physically allowed transformation of a quantum state, and every noise process can be modeled as one.

Examples:

- **Unitary evolution:** $\mathcal{E}(\rho) = U\rho U^\dagger$. A noise-free channel.
- **Measurement + discard outcome:** $\mathcal{E}(\rho) = \sum_k P_k \rho P_k$ where $\{P_k\}$ are projectors. The post-measurement state with the outcome forgotten.
- **Depolarizing channel:** $\mathcal{E}(\rho) = (1-p)\rho + p \cdot I/d$. Section 9.2.
- **Amplitude damping:** models energy loss $|1\rangle \to |0\rangle$. Section 9.3.

The Kraus Representation

Every quantum channel can be written as

$$\mathcal{E}(\rho) = \sum_k K_k \rho K_k^\dagger, \quad \sum_k K_k^\dagger K_k = I,$$

where the $\{K_k\}$ are **Kraus operators** (or "operation elements"). The Kraus representation is not unique — a channel has many equivalent Kraus decompositions — but any one of them fully specifies the channel.

Properties:

1. **A channel with one Kraus operator is a unitary** (up to global phase): $K = U$, so $\mathcal{E}(\rho) = U\rho U^\dagger$.
2. **A channel with two Kraus operators is a "one-error" channel**: probability p of applying an error, probability $1 - p$ of the identity. The depolarizing channel can be written with 4 Kraus operators (I, X, Y, Z).
3. **The number of Kraus operators is at most** d^2 for a d-dimensional system. So a single-qubit channel has at most 4 Kraus operators; a two-qubit channel has at most 16.
4. **The Kraus representation is a kind of "unraveling"** — it decomposes the channel into probabilistic branches, each branch being a distinct way the noise could act. This is the operational intuition.

Canonical Noise Channels

A handful of channels appear everywhere in quantum hardware modeling:

Bit flip channel

$$\mathcal{E}(\rho) = (1-p)\rho + p \cdot X\rho X.$$

Flips $|0\rangle \leftrightarrow |1\rangle$ with probability p. Classical bit error analogue.

Phase flip channel

$$\mathcal{E}(\rho) = (1-p)\rho + p \cdot Z\rho Z.$$

Flips the relative phase $|+\rangle \leftrightarrow |-\rangle$ with probability p. Purely quantum — no classical analogue.

Depolarizing channel

$$\mathcal{E}(\rho) = (1-p)\rho + p/3(X\rho X + Y\rho Y + Z\rho Z).$$

Applies a uniformly random Pauli error with probability p. The "worst-case" unbiased error model. Section 9.2.

Amplitude damping channel

$$\mathcal{E}(\rho) = E_0 \rho E_0^\dagger + E_1 \rho E_1^\dagger,$$

with $E_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{pmatrix}, E_1 = \begin{pmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{pmatrix}$.

Models energy relaxation: $|1\rangle \to |0\rangle$ with probability γ. The dominant error in superconducting qubits. Section 9.3.

Dephasing channel

$$\mathcal{E}(\rho) = \begin{pmatrix} \rho_{00} & \sqrt{1-\lambda}\rho_{01} \\ \sqrt{1-\lambda}\rho_{10} & \rho_{11} \end{pmatrix}.$$

Destroys coherence between $|0\rangle$ and $|1\rangle$ without changing populations. Closely related to the phase flip but continuous rather than probabilistic.

These five channels cover 95% of the noise models you'll encounter in practice. Real hardware is some weighted combination of these, with parameters (p, γ, λ) determined by calibration.

Stinespring Dilation

A deeper view: every quantum channel can be written as a unitary on a larger system.

Theorem (Stinespring 1955). For any CPTP channel $\mathcal{E}$ on a d-dimensional system, there exists an ancilla system of dimension at most d^2, a pure initial state $|0\rangle_{\text{anc}}$, and a unitary U on the joint system such that

$$\mathcal{E}(\rho) = \text{Tr}_{\text{anc}}\left[U(\rho \otimes |0\rangle\langle 0|_{\text{anc}})U^\dagger\right].$$

Interpretation. Every noise channel is equivalent to: add an ancilla in a known state, apply a joint unitary, then trace out (i.e., forget) the ancilla. The noise comes from the information lost when you discard the ancilla.

Why this matters. It means noise is not fundamentally mysterious — it is always "unitary evolution with part of the system hidden." This is important because:

1. It explains why quantum error correction works: if you can track the ancilla, the noise is reversible.
2. It unifies different noise models: they are all different choices of ancilla and joint unitary.
3. It connects to the fluctuation-dissipation theorem of statistical mechanics.

The Pauli Transfer Matrix

A useful computational tool: the **Pauli transfer matrix (PTM)** of a channel.

For a single-qubit channel, write $\rho = (I + r_x X + r_y Y + r_z Z)/2$ (Bloch vector representation). The channel acts linearly on $\mathbf{r} = (r_x, r_y, r_z)$:

$$\mathbf{r}' = T\mathbf{r} + \mathbf{t},$$

where T is a 3×3 matrix and $\mathbf{t}$ is a translation vector. Together $(T, \mathbf{t})$ form the **affine map** representation of the channel.

For the standard channels:

- **Identity:** $T = I$, $\mathbf{t} = 0$.
- **Bit flip with prob p:** $T = \text{diag}(1, 1-2p, 1-2p)$, $\mathbf{t} = 0$.
- **Depolarizing with prob p:** $T = (1-p)I$, $\mathbf{t} = 0$.
- **Amplitude damping with prob γ:** $T = \text{diag}(\sqrt{1-\gamma}, \sqrt{1-\gamma}, 1-\gamma)$, $\mathbf{t} = (0, 0, \gamma)$.

The PTM is convenient for computations — composition of channels is just matrix multiplication of PTMs, and expectation values of Pauli observables can be read off directly.

Composing Channels

Real circuits have many noise events. How do they compose?

Sequential composition. If you apply channels $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_L$ in sequence, the total channel is $\mathcal{E}_{\text{total}} = \mathcal{E}_L \circ \cdots \circ \mathcal{E}_2 \circ \mathcal{E}_1$.

In Kraus form: if $\mathcal{E}_i(\rho) = \sum_k K_{i,k} \rho K_{i,k}^\dagger$, then $\mathcal{E}_{\text{total}}$ has Kraus operators $K_{L,k_L} \cdots K_{2,k_2} K_{1,k_1}$ over all index sequences $(k_1, \ldots, k_L)$. The number of Kraus operators can grow exponentially — for an L-gate circuit with 2-Kraus-operator channels, up to 2^L terms.

In PTM form: $T_{\text{total}} = T_L \cdots T_2 T_1$. A single matrix product, regardless of circuit depth. This is much cheaper computationally.

For a depth-L circuit with uniform depolarizing channel $\mathcal{E}_p$ at each gate, the composed channel is $\mathcal{E}_{\text{total}}(\rho) = (1-p)^L \rho + (1 - (1-p)^L) \cdot I/d$. The **fidelity decays exponentially in** L — this is the root of all hardware-imposed depth limits.

Noise Rate Vs Fidelity

Two common ways of parameterizing noise:

1. Error probability p**.** The probability that a gate application produces an error. Used for discrete noise models (Pauli channels).

2. Process fidelity F**.** The fidelity of the noisy channel to the ideal channel. Related to p by (for depolarizing) $F = 1 - p(d^2-1)/d^2 \approx 1 - p$ for small p.

3. Average gate fidelity $\bar{F}$**.** The fidelity averaged over input states (Haar average). Related to process fidelity by $\bar{F} = (dF + 1)/(d+1)$.

For practical purposes, **gate fidelity** is the number you'll see on hardware spec sheets: "single-qubit fidelity 99.9%, two-qubit fidelity 99.5%". The corresponding error probabilities are 10^{-3} and 5×10^{-3} respectively.

In 2026, superconducting qubits typically achieve $\sim 99.9\%$ single-qubit and $\sim 99.5 - 99.8\%$ two-qubit fidelity. Trapped ions reach $\sim 99.99\%$ single-qubit and $\sim 99.9\%$ two-qubit. Neutral atoms are somewhere in between.

For a VQA circuit with hundreds of gates, these small per-gate errors compound: a 500-gate circuit at 99.5% two-qubit fidelity has total circuit fidelity $0.995^{500} \approx 0.08$ — only 8% of shots give correct results, 92% are pure noise.

Noise In The Heisenberg Picture

An alternative and often cleaner view: push the noise to the **observables**, not the states.

In the Heisenberg picture, a channel $\mathcal{E}$ acts on observables via its **adjoint** $\mathcal{E}^\dagger$:

$$\text{Tr}[O\mathcal{E}(\rho)] = \text{Tr}[\mathcal{E}^\dagger(O)\rho].$$

Implication for VQAs: instead of computing $\mathcal{E}(\rho)$ and then measuring O, you can compute $\mathcal{E}^\dagger(O)$ once and measure it against the noiseless state.

For depolarizing noise: $\mathcal{E}_p^\dagger(O) = (1-p)O + p \cdot \text{Tr}(O)/d$. The observable's coefficient shrinks by $(1-p)$ while the identity component is unaffected. For Pauli observables (traceless), this is just $O \to (1-p)O$ — the expectation value is scaled down by $(1-p)$.

Compound effect: after L noisy gates with depolarizing rate p, the effective observable is $(1-p)^L O$. Measuring this is the same as measuring O on the noiseless state but with the result scaled by $(1-p)^L$ — which is exactly the expected fidelity decay.

This Heisenberg-picture view is the foundation of **zero-noise extrapolation** (node 9.5) and other mitigation techniques — you're essentially rescaling the measurement to compensate for the noise's effect on observables.

How VQAs See Noise

When running a VQA on noisy hardware, noise affects several things:

1. The cost function becomes biased. $C_{\text{noisy}}(\boldsymbol{\theta}) = \text{Tr}[H\mathcal{E}(|\psi(\boldsymbol{\theta})\rangle\langle\psi(\boldsymbol{\theta})|)] \neq C_{\text{ideal}}(\boldsymbol{\theta})$. The optimizer is optimizing the wrong function.

2. The bias depends on $\boldsymbol{\theta}$. Different parameter values give different noisy costs, not all shifted by the same amount. So the optimizer can be actively misled.

3. The gradient becomes biased. $\nabla C_{\text{noisy}} \neq \nabla C_{\text{ideal}}$. Parameter-shift gradients are still *exact for the noisy cost*, but that's not the cost you want.

4. The landscape changes shape. Local minima, saddle points, and plateaus can all shift under noise. Module 4.5 covered the noise-induced plateau as an extreme example.

5. Shot noise adds on top. Even the noisy cost has to be estimated from finite shots, so there's an additional stochastic layer.

Practical upshot: the optimizer trained on a noisy VQA converges to the wrong minimum, and the wrongness grows with circuit depth and noise rate. This is the core practical problem that Modules 9-10 address.

Structural Role

This node is the **mathematical foundation** of Module 9. It introduces quantum channels, the Kraus and Stinespring representations, the Pauli transfer matrix, the composition rules, and the impact of noise on VQA cost functions.

It is the prerequisite for every other node in Module 9 — subsequent nodes (9.2 depolarizing, 9.3 amplitude damping, 9.4 mitigation, 9.5 ZNE, etc.) all build on the quantum-channel framework established here.

Relations to Other Concepts

- **Kraus (1971)** — the operator-sum representation.
- **Stinespring (1955)** — the dilation theorem.
- **Nielsen-Chuang** — *Quantum Computation and Quantum Information*, canonical reference.
- **Module 4.5 (Noise-induced plateaus)** — the qualitative consequence of these models.
- **Module 7.6 (Feedback control)** — treating noise as a disturbance to be rejected.

Worked Example — Depolarizing Channel On A 1-Qubit State

Take $\rho = |+\rangle\langle+| = (I+X)/2$ and apply the single-qubit depolarizing channel at rate $p = 0.1$:

$$\mathcal{E}_{0.1}(\rho) = 0.9 \cdot \rho + 0.1 \cdot I/2 = 0.9 \cdot (I+X)/2 + 0.1 \cdot I/2 = (I + 0.9X)/2.$$

The Bloch vector has shrunk from $(1, 0, 0)$ to $(0.9, 0, 0)$ — the state has moved toward the maximally mixed state.

Expectation value of X: $\mathrm{Tr}[X\rho] = 1$ (ideal) vs $\mathrm{Tr}[X\mathcal{E}(\rho)] = 0.9$ (noisy). The noise has decreased the expectation by a factor of $0.9 = 1 - p$.

Effect on a VQA cost function. If the cost is $C = \langle X \rangle$, the VQA sees $C_{\text{noisy}} = 0.9 C_{\text{ideal}}$. The optimizer still finds the right minimum (scaling doesn't change the location of extrema), just with a weaker signal. Signal-to-noise ratio is decreased, but convergence location is preserved.

Multi-layer circuit. Apply L layers of depolarizing noise. The expectation becomes $0.9^L \langle X \rangle$. For $L = 20$, this is $0.9^{20} \approx 0.12$ — only 12% of the signal remains. Below some threshold, the signal is indistinguishable from shot noise and the optimizer stalls.

This is the quantitative mechanism behind the depth limit on noisy hardware.

Visualization

Pending. A Bloch sphere showing a pure state and its trajectory under depolarizing noise as p increases from 0 to 1. The Bloch vector shrinks toward the center (maximally mixed). Caption: "Noise contracts the Bloch sphere. Everything collapses to $I/2$."

Exercises

1. Write down the Kraus operators for a bit flip channel with probability p. Verify they satisfy $\sum K^\dagger K = I$.
2. For a two-qubit system with single-qubit depolarizing noise (rate p) on each qubit independently, write down the joint channel as a Kraus decomposition. How many Kraus operators does it have?
3. (VQA) A VQA circuit has 200 gates with average two-qubit error $p = 0.005$. Estimate the total circuit fidelity. Is this sufficient for a useful result?

Research Extensions

- **Non-Markovian noise models** — correlations between errors that violate the CPTP assumption.
- **Learned noise models** — data-driven noise characterization from hardware calibration.
- **Structured noise for VQA design** — ansätze designed to exploit specific noise channels.
- **Coherent noise** — over-rotations and miscalibrations that don't fit the standard Pauli framework.

Cross-links to Other Courses

- **Open Quantum Systems** — Lindblad equation, master equations, decoherence theory.
- **Quantum Error Correction** — stabilizer codes, syndromes, logical operations.
- **Module 4.5 (Noise-induced plateaus)** — the qualitative effect on trainability.
- **Module 9.2 (Depolarizing Noise)** — the canonical example.

Depolarizing Noise

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.2

Depolarizing noise is the **workhorse noise model** of VQA theory. It is not the most physically realistic — real hardware has a mix of amplitude damping, dephasing, coherent errors, and crosstalk — but it is the most **mathematically tractable**, the most **symmetric**, and the most **commonly assumed** in theoretical analyses.

Understanding depolarizing noise deeply is worthwhile for several reasons:

1. It gives a **clean upper bound** on the effects of noise — if your VQA can handle depolarizing at rate p, it can usually handle more structured noise at similar rates.
2. It is the **simplest to simulate** — a single rate parameter controls everything.
3. It **reveals the depth-fidelity tradeoff** in closed form.
4. It is the **canonical example** for barren plateau theory under noise (Module 4.5, Wang et al. 2021).

This node derives the single-qubit depolarizing channel carefully, covers the multi-qubit generalizations, works through the composition rules for deep circuits, and explains why depolarizing noise is the "uniform" benchmark against which other noise models are measured.

The Single-Qubit Depolarizing Channel

The single-qubit depolarizing channel at rate p is defined as

$$\mathcal{D}_p(\rho) = (1-p)\rho + \frac{p}{3}\left(X\rho X + Y\rho Y + Z\rho Z\right).$$

Interpretation: with probability $1-p$, nothing happens (identity). With probability p, a uniformly random Pauli error X, Y, Z is applied. The factor $1/3$ splits the error probability evenly among the three Pauli types.

Kraus operators:

$$K_0 = \sqrt{1-p}\, I, \quad K_X = \sqrt{p/3}\, X, \quad K_Y = \sqrt{p/3}\, Y, \quad K_Z = \sqrt{p/3}\, Z.$$

Alternative form. Using the identity $(X\rho X + Y\rho Y + Z\rho Z + \rho)/4 = I/2 \cdot \mathrm{Tr}(\rho)$, the channel can be rewritten as

$$\mathcal{D}_p(\rho) = (1 - 4p/3)\rho + (4p/3) \cdot I/2.$$

Letting $\lambda = 4p/3$, this is $(1-\lambda)\rho + \lambda \cdot I/2$ — a **convex combination of the original state and the maximally mixed state**. This is the "depolarization" intuition: the state gets mixed with noise at rate λ.

Rescaling convention. Some authors use p to mean the total error probability (p above), others use p to mean the λ parameter. Be careful with the conventions.

Effect On Observables

Because the depolarizing channel is symmetric under Pauli conjugation, its action on Pauli observables is particularly simple:

$$\mathcal{D}_p^\dagger(P) = (1 - 4p/3)P \quad \text{for any non-identity Pauli } P.$$

And $\mathcal{D}_p^\dagger(I) = I$ (identity is preserved).

Consequence. Every Pauli expectation value is *uniformly shrunk* by a factor of $1 - 4p/3$. The measurement is unchanged in direction — only in magnitude.

For VQAs: if the cost Hamiltonian is $H = \sum_i c_i P_i$ (decomposed into Paulis), then the noisy cost is

$$C_{\text{noisy}}(\boldsymbol{\theta}) = (1 - 4p/3) \cdot C_{\text{ideal}}(\boldsymbol{\theta}) + 4p/3 \cdot \text{Tr}(H)/d.$$

The noisy cost is an **affine transformation** of the ideal cost: scaled by $(1 - 4p/3)$ and shifted by a constant. **Importantly, this affine transformation preserves the *location* of the minimum** — it only changes the value at the minimum and the slope of the landscape.

This is why depolarizing noise is often called "benign" for VQA optimization: the optimizer still finds the right parameters, just with a reduced signal-to-noise ratio. More problematic noise types (amplitude damping, coherent errors) actually *move* the minimum.

Multi-Qubit Depolarizing Channels

Several distinct generalizations exist for n qubits:

1. Independent single-qubit depolarizing

Each qubit experiences depolarizing noise independently:

$$\mathcal{D}_p^{\otimes n}(\rho).$$

Properties: factors over qubits; each single-qubit reduced state is depolarized independently; entanglement is *not* directly affected by the channel structure but does decay because of the per-qubit depolarization.

This is the most physically realistic model when the dominant noise is uncorrelated single-qubit error.

2. Global depolarizing

$$\mathcal{D}_p^{\text{global}}(\rho) = (1 - p)\rho + p \cdot I/2^n.$$

With probability p, the whole n-qubit state is replaced by the maximally mixed state; with probability $1 - p$, nothing happens.

Properties: the simplest theoretical model; often used in barren plateau proofs; rarely physically accurate, but gives clean bounds.

3. Correlated two-qubit depolarizing

Applied to pairs of qubits where two-qubit gates act:

$$\mathcal{D}_p^{(2)}(\rho) = (1 - p)\rho + \frac{p}{15} \sum_{k=1}^{15} P_k \rho P_k,$$

where the sum is over the 15 non-identity two-qubit Paulis. Used to model two-qubit gate errors.

Practical hardware modeling: real hardware is usually approximated as **single-qubit depolarizing after each single-qubit gate plus two-qubit depolarizing after each two-qubit gate**, with rates determined by the respective gate fidelities.

The Depth-Fidelity Tradeoff

A central calculation: **how does fidelity decay with circuit depth under depolarizing noise?**

For a circuit with L gates, each followed by depolarizing noise at rate p, the total channel on any Pauli observable is

$$\mathcal{D}^L(P) = (1 - 4p/3)^L P.$$

The signal decays exponentially in depth. Define the **effective depth** at which the signal is below the shot noise floor:

$$(1 - 4p/3)^{L^*} = \frac{1}{\sqrt{N_{\text{shots}}}}.$$

Solving: $L^* = -\log(\sqrt{N_{\text{shots}}})/\log(1 - 4p/3) \approx (3/4p) \cdot \log(\sqrt{N_{\text{shots}}})$.

Numerical examples:

- $p = 10^{-3}$, $N_{\text{shots}} = 10^6$: $L^* \approx 750 \cdot 6.9 \approx 5200$ gates.
- $p = 10^{-2}$, $N_{\text{shots}} = 10^6$: $L^* \approx 75 \cdot 6.9 \approx 520$ gates.
- $p = 10^{-1}$, $N_{\text{shots}} = 10^6$: $L^* \approx 7.5 \cdot 6.9 \approx 52$ gates.

Implication: for current hardware ($p \sim 5 \times 10^{-3}$ for two-qubit gates), useful VQA circuits are limited to ~1500 two-qubit gates. Beyond that, the signal is lost in noise and no amount of shots recovers it.

This is the **fundamental depth barrier** for NISQ-era VQAs, and it is tight — depolarizing noise is essentially the best-case scenario.

Depolarizing Noise And Barren Plateaus

Wang et al. (2021) showed that depolarizing noise creates **noise-induced barren plateaus** in addition to the unitary plateaus of Module 4.

Theorem (Wang et al. 2021). For a VQA with depth L under single-qubit depolarizing noise at rate p per gate, the gradient variance is bounded by

$$\text{Var}[\partial_j C_{\text{noisy}}] \leq C_0 \cdot (1 - 4p/3)^{2L},$$

where C_0 is a constant depending on the cost.

Implication: the gradient variance decays *exponentially in depth times noise rate*. Even ansätze that are trainable in the noiseless case become untrainable at sufficient depth under noise.

The noise-induced plateau depth: setting the variance bound equal to the shot noise floor $1/N_{\text{shots}}$, the plateau sets in at depth

$$L_{\text{NP}} \approx \frac{1}{4p/3} \cdot \frac{1}{2} \log(N_{\text{shots}} C_0^{-1}).$$

For $p = 5 \times 10^{-3}$, $N_{\text{shots}} = 10^6$, this is $L_{\text{NP}} \approx 500 - 1000$ gates — consistent with the depth limit from the fidelity argument.

The fundamental point: the depth limit from fidelity decay and the depth limit from noise-induced plateaus are *the same limit*, approached from two different angles. Depolarizing noise caps the usable circuit depth in exactly the regime where shot noise starts to dominate.

Why Depolarizing Is The Worst Case (In Some Sense)

A common claim: depolarizing noise is the "worst case" among unital noise channels at fixed average error rate.

Precise statement: for a fixed average gate fidelity, the depolarizing channel has the **maximum entropy** (most random) distribution of error types. Any structured noise (e.g., only phase errors, or only bit flips) gives a more concentrated error distribution.

Operational consequence: mitigation techniques that work for depolarizing noise work for more structured noise. Conversely, analyses that fail under depolarizing noise are unlikely to succeed under arbitrary noise.

Caveat: this is only true for **unital** channels (channels that preserve the maximally mixed state). **Non-unital** channels like amplitude damping are fundamentally different — they have a *preferred direction* in state space, and the "worst case" framing doesn't apply. Amplitude damping is discussed in node 9.3.

So: **depolarizing noise is the worst case among unital noise models**, and it is a useful upper bound for any noise that can be approximated as unital.

The Pauli Twirling Trick

A useful technique: **Pauli twirling** converts an arbitrary single-qubit channel into a depolarizing channel of equivalent fidelity.

Procedure. Before and after the noisy gate, apply a uniformly random Pauli (sampled from $\{I, X, Y, Z\}$). Average over many shots.

Effect. The average channel becomes

$$\bar{\mathcal{E}}(\rho) = \frac{1}{4} \sum_{P} P\mathcal{E}(P\rho P)P.$$

For a general single-qubit channel, this averages out to a **depolarizing channel** with the same average gate fidelity as the original.

Why this matters: twirling converts "messy" noise into "clean" depolarizing noise. Theoretical analyses that assume depolarizing noise are then *actually valid* for the twirled system, even if the raw hardware noise is more complex.

Cost: twirling doesn't reduce the noise — it just symmetrizes it. You pay for the simplification with the extra Pauli gates at each noise event.

Pauli twirling is the foundation of **randomized benchmarking** (the standard gate fidelity measurement protocol) and is used implicitly in many VQA mitigation schemes.

Depolarizing Noise In Practice

A few practical observations:

1. Real hardware is not purely depolarizing. Superconducting qubits have amplitude damping (energy loss) as the dominant noise; trapped ions have Rabi frequency errors; neutral atoms have loss and Rydberg decay. Depolarizing is an approximation.

2. The approximation is often good enough. For theoretical estimates and order-of-magnitude calculations, depolarizing is sufficient. Deviations matter at the factor-of-2 level, not the factor-of-100 level.

3. Simulation vs hardware. Depolarizing noise is the standard model in simulators (Qiskit Aer, Cirq, PennyLane). When you run a "noisy simulation," you're typically getting depolarizing unless you specify otherwise.

4. Randomized benchmarking returns depolarizing rates. The standard gate-fidelity measurement protocol assumes the noise is depolarizing (or has been twirled). Reported gate fidelities are depolarizing fidelities, which may differ from the "true" fidelity of the underlying physical noise.

5. Mitigation techniques built for depolarizing often extend. Zero-noise extrapolation (node 9.5) was originally developed for depolarizing noise and extends to broader noise classes. Similarly, probabilistic error cancellation is depolarizing-first.

So: depolarizing noise is both the theoretical default *and* the practical default, and knowing it deeply pays off.

Structural Role

This node is the **canonical noise example** of Module 9. It defines the single- and multi-qubit depolarizing channels, derives their action on observables, computes the depth-fidelity tradeoff explicitly, and establishes the connection to noise-induced plateau theory.

It is the benchmark against which other noise models (amplitude damping in 9.3, coherent errors in later sections) are compared.

Relations to Other Concepts

- **Nielsen-Chuang** — standard reference for the depolarizing channel.
- **Wang et al. (2021)** — noise-induced barren plateau theorem.
- **Randomized benchmarking (Magesan et al. 2011)** — fidelity measurement assuming depolarizing noise.
- **Pauli twirling** — converting arbitrary noise to depolarizing.
- **Module 4.5 (Noise-induced plateaus)** — the qualitative version.

Worked Example — Depth Limit For A VQA On 2026 Hardware

Consider a 20-qubit VQE on a chemistry Hamiltonian, running on superconducting hardware with: - Single-qubit gate fidelity: 99.9% $\to p_1 = 10^{-3}$. - Two-qubit gate fidelity: 99.5% $\to p_2 = 5 \times 10^{-3}$. - Shot budget: 10^6 per cost evaluation.

Ansatz: Hardware-Efficient Ansatz with 10 layers, each layer having 20 single-qubit gates + 19 two-qubit gates = 39 gates/layer. Total: 390 gates per circuit, of which 200 are two-qubit.

Total error: approximating each gate as an independent depolarizing channel, the circuit fidelity is

$$F \approx (0.999)^{200} \cdot (0.995)^{200} = 0.819 \cdot 0.367 = 0.30.$$

Only 30% of the signal survives. Combined with shot noise at $1/\sqrt{10^6} = 10^{-3}$, the effective SNR is $0.30/10^{-3} = 300$ — good enough for training.

Doubled depth (20 layers): $F \approx 0.30^2 = 0.09$. SNR ≈ 90. Still workable but degraded.

Quadrupled depth (40 layers): $F \approx 0.30^4 = 0.008$. SNR ≈ 8. Near the noise floor.

Conclusion: for this hardware, 10-20 layer circuits are usable, 40+ layers are beyond the noise-limited depth. This is consistent with the state-of-the-art 2026 experimental VQE demonstrations, which typically use 5-20 layer ansätze.

Visualization

Pending. A log-scale plot of gate fidelity vs circuit depth for several values of p (from 10^{-4} to 10^{-2}) showing the exponential decay. The shot noise floor is a horizontal line; the intersections give the maximum usable depth. Caption: "Depolarizing noise sets the NISQ depth ceiling."

Exercises

1. Compute the Pauli transfer matrix of a single-qubit depolarizing channel at rate p.
2. Show that the composition of two depolarizing channels at rates p_1, p_2 is a depolarizing channel. What is its rate?
3. (VQA) For a 30-qubit VQA with 500 two-qubit gates at fidelity 99.8%, estimate the circuit fidelity and the shot budget needed for a useful cost estimate.

Research Extensions

- **Non-uniform depolarizing rates** — different rates for different gates or qubits.
- **Coherent depolarizing deviations** — what if the noise is almost-depolarizing but not exactly?
- **Active depolarization** — intentionally adding depolarizing noise as a form of regularization for training stability.
- **Depolarizing-equivalent circuit compilation** — transforming circuits so their effective noise is depolarizing.

Cross-links to Other Courses

- **Open Quantum Systems** — Lindblad master equations for depolarizing noise.
- **Randomized Benchmarking** — the standard gate fidelity protocol.
- **Module 4.5 (Noise-induced plateaus)** — the qualitative version.
- **Module 9.5 (Zero-noise extrapolation)** — the main mitigation technique for depolarizing noise.

Gate Errors and Coherence Times

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.3

Node 9.2 treated depolarizing noise as the canonical model. This node gets **physical**: it explains where noise actually comes from in real hardware, what "gate error" and "coherence time" mean operationally, and how the abstract quantum channels of 9.1 connect to the physical parameters T_1, T_2, gate duration, and fidelity that appear on hardware spec sheets.

The goal: after this node, you can read a hardware specification (e.g., "IBM Heron r1 processor, 133 qubits, median T_1 150 µs, median T_2 80 µs, median 2Q fidelity 99.5%") and know exactly what it implies for VQA depth, shot budget, and expected performance.

This is the **bridge between theory and practice** — the point where abstract channels become concrete numbers.

The Sources Of Noise

Real quantum hardware has several distinct noise sources, each with its own physics and its own channel representation.

1. Energy relaxation (amplitude damping)

Physics: the excited state $|1\rangle$ can decay to the ground state $|0\rangle$ via spontaneous emission (superconducting), photon scattering (ions, atoms), or other dissipative mechanisms. The rate is characterized by the **energy relaxation time** T_1.

Channel. The amplitude damping channel at rate $\gamma = 1 - e^{-t/T_1}$ (where t is the elapsed time):

$$\mathcal{A}_\gamma(\rho) = E_0 \rho E_0^\dagger + E_1 \rho E_1^\dagger,$$

with

$$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{pmatrix}, \quad E_1 = \begin{pmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{pmatrix}.$$

Key properties: - **Non-unital:** amplitude damping moves states *toward* $|0\rangle$, not toward $I/2$. There is a preferred direction. - **Breaks symmetry:** a state initialized to $|1\rangle$ decays, but $|0\rangle$ does not. Noise depends on the state. - **Cumulative:** at time t, the channel parameter is $\gamma(t) = 1 - e^{-t/T_1}$.

2. Dephasing (phase damping)

Physics: the coherence between $|0\rangle$ and $|1\rangle$ is destroyed by fluctuations in the environment (charge noise, flux noise, magnetic field variations). The rate is T_ϕ, and combines with T_1 to give the **total coherence time** T_2:

$$\frac{1}{T_2} = \frac{1}{2T_1} + \frac{1}{T_\phi}.$$

Channel. The phase damping channel at rate $\lambda = 1 - e^{-t/T_\phi}$:

$$\mathcal{P}_\lambda(\rho) = \begin{pmatrix} \rho_{00} & \sqrt{1-\lambda}\rho_{01} \\ \sqrt{1-\lambda}\rho_{10} & \rho_{11} \end{pmatrix}.$$

Key properties: - **Unital:** it preserves $I/2$. - **Diagonal-preserving:** computational basis populations are unchanged. - **Eigenvalue structure:** exponential decay of off-diagonal elements (coherences).

3. Coherent errors

Physics: miscalibrations in the control pulses cause systematic over- or under-rotations. Not random — deterministic but unknown.

Channel (example: over-rotation). $U_{\text{actual}} = R_Y(\theta + \delta)$ instead of $R_Y(\theta)$. No probability mixing — just a wrong unitary.

Key properties: - **Unitary:** still a valid quantum operation. - **Coherent accumulation:** many small errors can add up to a large error if they are correlated. - **Not captured by Pauli channels:** the standard depolarizing model misses coherent errors, which is a frequent source of theory-experiment discrepancy.

4. Crosstalk

Physics: a gate on qubit A affects qubit B due to unintended couplings. Purely a multi-qubit effect.

Channel. The ideal single-qubit gate U_A is replaced by a correlated two-qubit channel that depends on the state of qubit B. Hard to characterize, hard to model.

Key properties: - **Non-local:** creates correlations between distant errors. - **Context-dependent:** the effect depends on what other gates are happening in parallel. - **One of the dominant error sources** in 2026 superconducting hardware.

5. Measurement errors

Physics: the measurement operation itself is imperfect. A qubit in state $|0\rangle$ may be reported as "1" with some probability, and vice versa.

Channel. Modeled as a classical confusion matrix applied to the measurement outcome:

$$\begin{pmatrix} P(0_{\text{reported}}|0_{\text{true}}) & P(0_{\text{reported}}|1_{\text{true}}) \\ P(1_{\text{reported}}|0_{\text{true}}) & P(1_{\text{reported}}|1_{\text{true}}) \end{pmatrix}.$$

Typical values: 1-5% error rate for current superconducting hardware, $\sim 0.5\%$ for trapped ions.

Key properties: - **Classical:** can be corrected by post-processing if the confusion matrix is known. - **Asymmetric:** typically different rates for $0 \to 1$ and $1 \to 0$ errors. - **Dominates at shallow depth:** for very shallow circuits, measurement error can be the largest contribution.

The Relaxation-Time Parameters

The headline numbers for any piece of quantum hardware:

T_1 **(energy relaxation time):** the characteristic timescale for the excited state to decay. Typical values: - Superconducting (transmon): 50-300 µs. - Trapped ion: seconds. - Neutral atom (Rydberg): hundreds of µs. - Photonic: effectively infinite (no relaxation). - Silicon spin: milliseconds.

T_2 **(total dephasing time):** the characteristic timescale for coherences to decay, always $\leq 2T_1$. Typical values: - Superconducting: 20-200 µs (often $T_2 \ll 2T_1$ because of dephasing). - Trapped ion: hundreds of ms to seconds. - Neutral atom: 0.5-5 ms. - Photonic: effectively infinite. - Silicon spin: hundreds of µs (isotope-enriched).

T_2^* **(pure dephasing time without echo):** the dephasing time measured by a Ramsey experiment (no echo pulse). Usually shorter than T_2.

Gate time: the duration of a gate operation. Typical values: - Superconducting single-qubit: 20-50 ns. - Superconducting two-qubit: 100-500 ns. - Trapped ion: 10-500 µs. - Neutral atom: 100 ns-1 µs.

The gate error budget: a gate of duration τ on a qubit with coherence time T_2 has coherence-limited error $\sim \tau/T_2$. For superconducting hardware with $\tau = 300$ ns and $T_2 = 100$ µs, this is 3×10^{-3} — a ∼0.3% error floor from coherence alone. Actual two-qubit errors are usually 2-3x larger due to calibration and other non-idealities.

The Circuit Time Budget

For a circuit with L gates of duration τ, the total circuit time is $L\tau$. For the circuit to be useful, this must be much less than T_2:

$$L\tau \ll T_2 \quad \Longleftrightarrow \quad L \ll T_2/\tau.$$

Examples:

- Superconducting ($T_2 = 100$ µs, $\tau = 300$ ns): $L \ll 333$ gates. Usable depth ~ 100 gates.
- Trapped ions ($T_2 = 1$ s, $\tau = 100$ µs): $L \ll 10^4$ gates. Usable depth $\sim 1000 - 3000$ gates.
- Neutral atoms ($T_2 = 2$ ms, $\tau = 500$ ns): $L \ll 4000$ gates.

Observation: the gate-count-per-coherence-time is surprisingly similar across technologies in the low hundreds. The different technologies have different absolute speeds but similar "relative" circuit budgets.

For VQAs: the usable depth $\sim 100 - 1000$ gates is consistent with the depolarizing analysis of 9.2. Real depth is more like $\sim 10 - 50$ layers of an HEA for 20-30 qubits — limited by both coherence and noise-induced plateaus.

Fidelity As An Operational Metric

Gate fidelity is the headline metric, but it can mean several things:

1. Average gate fidelity $\bar{F}$: the average over Haar-random input states of the fidelity between the ideal and actual output. The number reported on spec sheets.

2. Process fidelity F_{proc}: the overlap between the ideal and actual process, normalized. Related to average fidelity by $\bar{F} = (dF_{\text{proc}} + 1)/(d + 1)$.

3. Entanglement fidelity: for two-qubit gates, the fidelity when the input is a maximally entangled state. Sometimes more informative.

4. Benchmarking fidelity: what **randomized benchmarking** reports. This is a specific operational protocol that effectively measures the twirled depolarizing rate, which is close to but not identical with the average gate fidelity.

For most practical purposes, **the randomized benchmarking fidelity is what hardware vendors report**, and that's what you should use in your depth-fidelity calculations. It's the depolarizing-equivalent fidelity after Pauli twirling.

The Amplitude Damping Vs Depolarizing Comparison

A practically important comparison: how different is amplitude damping from depolarizing?

At low rates ($\gamma \ll 1$): very similar. Both are approximately "error with probability $\sim \gamma$," and the detailed structure doesn't matter much for shallow circuits.

At higher rates: amplitude damping is *preferentially* toward $|0\rangle$, so it biases expectation values of Z toward $+1$ (the $|0\rangle$ eigenvalue). Depolarizing biases all Pauli expectations toward zero symmetrically.

For VQAs: if your cost function has significant Z character, amplitude damping will shift the cost systematically (upward for Z with positive coefficient). The optimizer will find a *different* minimum than the ideal one. This is a real practical issue with superconducting hardware.

Quantitative comparison: at average gate fidelity 99.5%, depolarizing noise shifts $\langle Z\rangle$ to $0.995\langle Z\rangle$; amplitude damping shifts $\langle Z\rangle$ to $\langle Z\rangle + (\text{const}) \cdot (1 - \text{rate})$ — a state-dependent additive shift.

Upshot: for theoretical analysis, assume depolarizing for clean bounds. For real hardware, use amplitude damping + dephasing. For quick practical estimates, depolarizing is usually accurate enough.

The Coherence-Limited Regime

A useful conceptual division: a VQA circuit is either **coherence-limited** or **gate-error-limited**.

Coherence-limited: the total circuit time is close to T_2, so the whole state decays before the circuit finishes. Further reductions in per-gate error don't help — the state is already decohering. This is the regime for ions (where individual gate fidelities are excellent but circuits are slow).

Gate-error-limited: the circuit is fast compared to T_2, but individual gates have non-trivial error rates. The total error is dominated by summing per-gate errors. This is the regime for superconducting qubits (fast gates, shorter coherence).

For VQA design:

- In the coherence-limited regime, **reducing gate count** is the right strategy. Every gate fewer is better because it saves circuit time.
- In the gate-error-limited regime, **reducing per-gate error** is the right strategy. Every calibration improvement compounds.

Most 2026 superconducting hardware is gate-error-limited for shallow circuits and coherence-limited for deep ones. The crossover depth depends on the specific machine.

Calibration And Drift

Real hardware parameters **drift** on timescales of minutes to hours:

- **Qubit frequencies** drift by tens of kHz over hours.
- **Gate fidelities** drift by 10-20% (relative) over a day.
- T_1, T_2 can vary 2x over a week.

Hardware providers run periodic **recalibration** (typically every 1-4 hours) to keep parameters within spec. Between calibrations, the noise model is slowly changing.

For VQAs: if a VQA training run takes 12 hours, the noise model at the end of the run is different from the model at the beginning. A static noise-aware compilation won't be valid throughout. The right response is **adaptive noise characterization** — periodically re-measuring the noise parameters and updating the compilation or the optimizer accordingly.

This is one of the practical deployment challenges that Module 10 (Adaptive Control Systems) addresses.

Structural Role

This node is the **physical-hardware** node of Module 9. It connects the abstract noise channels of 9.1-9.2 to the concrete parameters (T_1, T_2, gate time, fidelity) that appear on hardware spec sheets, and gives the practitioner the information needed to translate between hardware specs and VQA performance estimates.

It is the prerequisite for nodes 9.4 (mitigation) and 9.6 (noise-adaptive ansatz design), which build on the physical understanding.

Relations to Other Concepts

- **Bloch (1946), Redfield (1957)** — foundational work on relaxation theory.
- **Randomized benchmarking (Magesan et al. 2011)** — the standard fidelity measurement.
- **Quantum process tomography** — full characterization of a gate.
- **Module 9.1 (Quantum noise channels)** — the mathematical framework.
- **Module 10 (Adaptive control)** — handling drift via adaptation.

Worked Example — Noise Budget For A 15-Qubit QAOA

Target hardware: superconducting, $T_1 = 150$ µs, $T_2 = 80$ µs, single-qubit gate time 30 ns, two-qubit gate time 300 ns, single-qubit fidelity 99.9%, two-qubit fidelity 99.5%.

Circuit: 15-qubit QAOA at depth $p = 3$. Each layer has 15 single-qubit gates + 14 two-qubit gates = 29 gates. Three layers = 87 gates, of which 42 are two-qubit.

Time budget: 87 gates times average time ~ 200 ns ≈ 17 µs. **Well within $T_2 = 80$ µs.** Coherence is not the bottleneck.

Fidelity budget: - Single-qubit error contribution: $45 \cdot 10^{-3} = 0.045$. - Two-qubit error contribution: $42 \cdot 5 \times 10^{-3} = 0.21$. - Total depolarizing error: ≈ 0.26. Circuit fidelity ≈ 0.74.

Measurement error: typically 2% per qubit, so 15 qubits of measurement error contribute another ~ 0.3 to the total error.

Net: about 40-50% of the signal survives. Shots needed to resolve signal above shot noise: for a cost that is ~ 0.5 in magnitude on a good run and 0.25 after noise, shot noise of 10^{-3} (i.e., 10^6 shots) is comfortably below the signal.

Conclusion: this QAOA is workable. For depth 6, we'd double the gate count and the signal fidelity would drop to ~ 0.3. At depth 10, ~ 0.1 — workable but getting marginal. At depth 20, signal is lost.

This is the kind of calculation every experimentalist does before running a VQA.

Visualization

Pending. A chart showing, for several hardware platforms (superconducting, trapped ion, neutral atom, silicon), the typical T_1, T_2, gate time, and gate error. Overlaid: the usable circuit depth for each platform. Caption: "Hardware comes in flavors. Each flavor has a different depth budget."

Exercises

1. A qubit has $T_1 = 100$ µs, $T_2 = 60$ µs. What is the pure dephasing time T_ϕ?

2. For a superconducting two-qubit gate of duration 400 ns and fidelity 99.2%, what fraction of the error is coherence-limited vs gate-error-limited?
3. (VQA) For a 30-qubit VQE on trapped-ion hardware ($T_2 = 1$ s, two-qubit gate time 300 µs, fidelity 99.9%), estimate the maximum circuit depth.

Research Extensions

- **Time-dependent noise models** — capturing drift between calibrations.
- **Crosstalk characterization** — systematic methods for identifying and mitigating crosstalk.
- **Coherent-error-aware compilation** — reducing coherent errors via gate-sequence design.
- **Noise tomography** — full process characterization of hardware gates.

Cross-links to Other Courses

- **Open Quantum Systems** — Lindblad master equations.
- **Quantum Measurement Theory** — measurement errors and corrections.
- **Hardware Physics (superconducting/ion/atom)** — the underlying physics of specific platforms.
- **Module 9.4 (Noise mitigation)** — the response to the errors characterized here.

Noise Mitigation Techniques

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.4

Noise mitigation is the collection of **post-processing techniques** that reduce the effect of noise on the *expectation value* you extract from a noisy quantum circuit — without doing full fault-tolerant quantum error correction. The distinction is critical:

- **Error correction** (QEC) fixes the underlying *quantum state* using redundant encoding and active syndrome measurement. It is the long-term answer. It requires thousands of physical qubits per logical qubit and is not yet practical.
- **Error mitigation** does not fix the state. It uses classical post-processing to produce a better *estimate* of the noiseless expectation value from noisy runs. It is the NISQ-era answer.

Mitigation techniques pay for their accuracy with **sample complexity** — more circuit runs, not more qubits. This matches the VQA resource profile well, where qubits are scarce but classical compute is cheap.

This node surveys the major mitigation techniques: zero-noise extrapolation (ZNE), probabilistic error cancellation (PEC), readout error correction, Clifford data regression, symmetry verification, dynamical decoupling, and virtual distillation. Node 9.5 goes deep on ZNE specifically. Node 9.6 covers noise-adaptive ansatz design, which is a structurally different mitigation strategy.

What Mitigation Can And Cannot Do

Can:

1. **Estimate the noiseless expectation value** $\langle O \rangle_{\text{ideal}}$ with reduced bias, at the cost of increased variance.
2. **Remove systematic error** from specific known noise sources (e.g., readout errors, coherent calibration errors).
3. **Exploit symmetries** to reject errors that break a conserved quantity.
4. **Extend the usable depth** of a circuit by a factor of 2-5x under typical NISQ conditions.

Cannot:

1. **Recover the full quantum state** — you only get expectation values, not wavefunctions.
2. **Work past a fundamental sampling floor** — the sample complexity grows exponentially in the effective noise, so eventually you can't afford the shots.
3. **Fix errors that are indistinguishable from the signal** — e.g., if noise has the same symmetry structure as the observable.
4. **Replace error correction** for deep circuits — the overhead makes deep circuits unreachable.

Mitigation is the **right tool for the NISQ regime**: shallow-to-moderate circuits, moderate noise, tolerance for increased shot count. Outside that regime (either very shallow or very deep), different tools apply.

The Sample-Complexity Cost

A universal fact about mitigation: the reduction in bias comes with an increase in variance. Formally, if a mitigation technique multiplies the estimator by a factor $\gamma > 1$ (to undo noise shrinkage), the estimator variance grows by γ^2, and the required shot count to reach fixed precision grows the same way.

For modest noise ($p \sim 10^{-3}$ per gate, depth ~ 100), typical γ values are 2-10, meaning 4-100x more shots. Manageable.

For large noise ($p \sim 10^{-2}$, depth ~ 500), γ can blow up to 10^3-10^4, meaning 10^6-10^8x more shots. Unreachable.

The practical sweet spot: NISQ circuits with gate fidelity 99.5%-99.9% and depth 50-200 gates. Mitigation works here; it fails outside.

This sample-complexity barrier is why mitigation is not a long-term answer. It's a bridge to fault-tolerance.

Readout Error Correction

The simplest and most impactful mitigation. Measurement errors on current hardware are typically 1%-5% per qubit — much larger than gate errors. Luckily, they are also the easiest to characterize and correct.

Procedure:

1. **Characterize** the confusion matrix M, where M_{ij} is the probability of observing outcome i given the true outcome was j. This is done by preparing each computational basis state and measuring, typically with a few thousand shots.
2. **Invert** the confusion matrix: M^{-1} maps observed probabilities back to ideal ones.
3. **Apply** M^{-1} to the observed probability vector to recover the corrected distribution.

For n qubits, the full confusion matrix is $2^n \times 2^n$ — prohibitive beyond ~ 10 qubits. Practical variants:

- **Tensor product assumption:** treat each qubit's readout error independently. The confusion matrix factors as $M = M_1 \otimes M_2 \otimes \cdots \otimes M_n$. Characterization is $O(n)$ and inversion is $O(n)$.
- **Iterative Bayesian unfolding:** avoid explicit matrix inversion; iteratively refine the distribution using Bayes' rule. Handles correlated readout errors.
- **M3 (Matrix-free Measurement Mitigation):** solves a linear system to mitigate without forming the matrix. Scales to hundreds of qubits (Nation et al. 2021).

Effectiveness: removes most readout error bias at modest cost. Should be applied to *all* VQA experiments — it's essentially free relative to the underlying circuits.

Zero-Noise Extrapolation (ZNE) — Brief

Core idea: run the circuit at several artificially amplified noise levels, fit the expectation value as a function of noise strength, and extrapolate to zero noise.

Procedure:

1. Define a noise-amplification method (unitary folding, pulse stretching, or identity insertion) that scales the effective noise by factor $\lambda \in \{1, \lambda_2, \lambda_3, \ldots\}$.
2. Run the circuit at each amplification level and estimate $\langle O \rangle(\lambda)$.
3. Fit a curve (linear, exponential, Richardson) to $\langle O \rangle$ vs λ.
4. Extrapolate to $\lambda = 0$ to get the estimated noiseless value.

Pros: requires no detailed noise model, easy to implement, works with any observable.

Cons: extrapolation is extrapolation — the fit can be wrong, especially with large noise or non-monotonic curves. Typical accuracy improvement is 3-10x for NISQ regimes.

Node 9.5 goes into full detail: folding methods, fit choices, error analysis, practical recipes.

Probabilistic Error Cancellation (PEC)

A more ambitious technique: **invert the noise channel directly** via a quasi-probability decomposition.

Idea. Suppose the noise channel $\mathcal{N}$ is known and invertible. Then $\mathcal{N}^{-1}$ is a (quasi-)map: it can be written as a linear combination of physically implementable channels, possibly with negative coefficients:

$$\mathcal{N}^{-1} = \sum_k c_k \mathcal{E}_k, \quad \sum_k c_k = 1, \quad c_k \in \mathbb{R}.$$

Procedure. At each circuit run, sample a k with probability $|c_k|/\gamma$ (where $\gamma = \sum_k |c_k| \geq 1$ is the one-norm), apply the $\mathcal{E}_k$ operation, and multiply the measurement outcome by $\text{sign}(c_k) \cdot \gamma$.

Averaging many such runs produces an unbiased estimator of the noiseless expectation value.

The cost. The variance blowup factor is γ^2, and γ grows exponentially in the circuit depth for most realistic noise ($\gamma \sim e^{c \cdot L \cdot p}$ for noise rate p over depth L). So PEC works for shallow circuits but becomes prohibitively expensive for deep ones.

Pros: unbiased (not an extrapolation), gives mathematically rigorous guarantees.

Cons: requires precise knowledge of the noise channel (often unavailable), has exponential sample overhead, and the "quasi-probability" implementation is operationally awkward.

Status in 2026: PEC works on gate counts of 50-200 for superconducting hardware. Above that, sample complexity kills it. It is typically combined with ZNE (PEC for the short-range errors, ZNE for what remains).

Reference: Temme, Bravyi, Gambetta (2017) introduced PEC. Subsequent work by Takagi et al. (2022) gave sample complexity lower bounds showing the exponential is fundamental.

Clifford Data Regression (CDR)

A clever mitigation strategy using the fact that Clifford circuits are **classically simulable**.

Procedure.

1. Take your VQA circuit $U(\boldsymbol{\theta})$.
2. Construct a set of "nearby" Clifford circuits by replacing non-Clifford gates with their closest Clifford approximants.
3. Classically simulate each Clifford circuit to get the ideal expectation values $\{C_i^{\text{ideal}}\}$.
4. Run each Clifford circuit on the noisy hardware to get $\{C_i^{\text{noisy}}\}$.
5. Fit a regression model $f : C_i^{\text{noisy}} \mapsto C_i^{\text{ideal}}$ — typically linear.
6. Apply f to the noisy expectation value from the original (non-Clifford) circuit.

Intuition. The noise process is (approximately) consistent across Clifford and non-Clifford circuits — if you know how noise distorts the Cliffords, you can invert it for the non-Cliffords.

Pros: doesn't require a noise model, handles *systematic* bias in a principled way, often works when simple ZNE fails.

Cons: needs classical simulation of Clifford circuits (fine for moderate qubit counts; up to ~ 100 qubits for $\mathcal{O}(100)$-depth Cliffords), the "nearby Clifford" construction is heuristic, the regression can overfit.

Reference: Czarnik, Arrasmith, Coles, Cincio (2020) introduced CDR. Later extended to include polynomial regression and neural network models.

Practical note: CDR is the technique that most often outperforms ZNE on realistic VQA problems in the 50-qubit range. It should be considered a first-line mitigation alongside ZNE.

Symmetry Verification

Many physical Hamiltonians have **conserved quantities** — particle number, spin parity, total charge. If the ideal quantum circuit preserves these symmetries, then measurement outcomes that violate them are *known to be wrong* and can be discarded.

Procedure.

1. Identify a conserved quantity Q (typically a Pauli sum that commutes with the cost Hamiltonian).
2. Measure Q (either as part of the cost estimation or via a separate ancilla-based check).
3. Discard any shot where Q does not match the expected value.
4. Use only the remaining shots to estimate the cost.

Example (chemistry VQE). For a molecule with N electrons, the particle number $\hat{N} = \sum_i \hat{n}_i$ is conserved. Discard any Fock-space outcome with $\hat{N} \neq N$. This removes errors that change particle number (which amplitude damping does).

Pros: cheap (no extra circuits), principled (error rejection is unbiased), removes a physically meaningful class of errors.

Cons: only works if the symmetry is actually preserved by the circuit (not all ansätze preserve particle number), reduces the *effective* shot count (shots violating the symmetry are discarded), doesn't help with errors that preserve the symmetry.

Reference: McArdle, Yuan, Benjamin (2019) developed symmetry verification for VQE. It's now standard in chemistry VQA pipelines.

Dynamical Decoupling (DD)

A mitigation technique at the **pulse level** rather than the circuit level. It reduces coherent and low-frequency noise by rapid unitary "refocusing" pulses.

Intuition. If you're sitting idle on a qubit for time τ and there's a slowly varying environment field, the qubit acquires a phase error. But if you apply an X pulse halfway through, the phase error accumulated in the first half gets reversed in the second half — the net error is small.

Standard DD sequences.

- **Hahn echo:** a single π pulse in the middle. Cancels static inhomogeneous dephasing.
- **CPMG:** a train of π pulses at regular intervals. Cancels frequency noise up to a higher frequency.
- **XY-4, XY-8:** pulse sequences that protect against more general noise (not just dephasing).
- **Uhrig DD:** non-uniform pulse spacing optimized for specific noise spectra.

Application to VQAs. In a VQA, during the "dead time" when qubits are idle (waiting for other gates to finish, or during measurement of other qubits), DD pulses can be inserted to refocus errors. This is especially useful for superconducting qubits with long dead times.

Pros: reduces coherent and low-frequency noise at pulse level, no computational overhead, transparent to the circuit (the DD pulses are inserted into the idle periods).

Cons: doesn't help with high-frequency (Markovian) noise — and most gate errors *are* Markovian. Primarily helps with idle errors.

Reference: Viola, Lloyd (1998) introduced DD theoretically. Modern usage in VQAs: Pokharel et al. (2018) showed 3-5x fidelity improvement on IBM hardware via DD insertion.

Virtual Distillation

A more recent (2020+) technique that uses **multiple copies** of the noisy state to purify it.

Idea. If the noisy state is $\rho = (1-\epsilon)|\psi\rangle\langle\psi| + \epsilon\sigma$ with $\epsilon \ll 1$, then $\rho^2/\mathrm{Tr}(\rho^2)$ is closer to $|\psi\rangle\langle\psi|$ than ρ is. Higher powers ρ^M are closer still.

Procedure.

1. Prepare M copies of the noisy state.
2. Use a circuit that implements the "derangement" operation on the copies.
3. Measure an observable O along with the derangement.
4. The combined measurement gives an unbiased estimator of $\mathrm{Tr}(\rho^M O)/\mathrm{Tr}(\rho^M)$ — which is close to $\langle\psi|O|\psi\rangle$.

Pros: no noise model required, exponential suppression in M for the coherent error part.

Cons: requires M copies of the state (so $M\times$ qubit count), the derangement circuit introduces its own noise, the method is biased in a specific way that needs correction.

Reference: Huggins et al. (2021), Koczor (2021) independently introduced virtual distillation. Still somewhat research-stage; not yet standard in VQA pipelines but is the main 2024-2026 research direction for error mitigation.

Mitigation In Practice: A Decision Tree

For VQA practitioners, the practical question is: *which* mitigation technique should I apply?

1. **Always apply readout error correction.** Cost is negligible, benefit is significant. Non-negotiable.
2. **If your Hamiltonian has a conserved symmetry, apply symmetry verification.** Also cheap and principled.
3. **If your hardware has long idle times, apply dynamical decoupling.** Costs nothing in shots; reduces idle errors.
4. **For the "main" mitigation**, choose between ZNE and CDR based on depth:
   - Shallow ($L < 100$): ZNE with linear or exponential fit. Simple and effective.
   - Moderate depth ($100 < L < 300$): CDR often outperforms ZNE. Use if you can classically simulate the Cliffords.
   - Deep ($L > 300$): Neither works well. Consider reducing depth or switching to a shallower ansatz.
5. **For research-grade VQAs**, consider PEC (if noise model is known) or virtual distillation (if qubit count allows). Higher effort, can give best-in-class results.

Combining techniques. Readout correction, symmetry verification, and DD can be stacked with ZNE or CDR. Be careful about combining PEC with anything else — the quasi-probability math gets messy.

Mitigation And The Thesis

The thesis view of mitigation:

1. **Mitigation is a form of classical side-computation applied to the noisy optimizer.** It is not part of the quantum dynamics — it is outer-loop post-processing.
2. **HOPSO's structure makes mitigation cleaner.** Because HOPSO separates the dynamical update from the objective generator, mitigation can be applied to the objective generator (which produces C_{noisy}) without touching the optimizer dynamics. This is architecturally clean.
3. **Mitigation reduces bias but not the fundamental signal-to-noise tradeoff.** The usable depth is set by the fidelity budget plus the shot budget, and mitigation just shuffles the budget between the two. No free lunch — the question is whether your VQA can afford the tradeoff.
4. **The future of mitigation is integration with optimization.** Mitigation techniques applied in *series* (observe, correct, optimize) are one architecture; applied in *parallel* (the optimizer is aware of the mitigation cost and adjusts accordingly) is another. Module 10 takes up the latter.

Structural Role

This node is the **survey of the mitigation toolkit** for Module 9. It introduces each major mitigation technique, describes what it does and its cost, and gives a practical decision tree for when to use each.

It is the parent node for 9.5 (ZNE in depth) and 9.6 (noise-adaptive ansatz design). Together, these three nodes cover the main lines of attack against NISQ noise for VQAs.

Relations to Other Concepts

- **Temme, Bravyi, Gambetta (2017)** — PEC original.
- **Li, Benjamin (2017)** — ZNE original.
- **Czarnik et al. (2020)** — CDR original.
- **McArdle et al. (2019)** — symmetry verification for VQE.
- **Huggins et al. (2021), Koczor (2021)** — virtual distillation.
- **Viola, Lloyd (1998)** — DD theory.
- **Nation et al. (2021)** — M3 readout mitigation.
- **Cai et al. (2023)** — review of quantum error mitigation.

Worked Example — Mitigation Stack For A 10-Qubit VQE

Consider a 10-qubit VQE for a chemistry problem, running on superconducting hardware with: - Two-qubit gate fidelity 99.5%. - Readout fidelity 97%. - Circuit depth $L = 80$ (including 40 two-qubit gates). - Particle number conservation (electrons fixed at 5). - 10^5 shots budget per cost evaluation.

Without mitigation. - Circuit fidelity: $(0.995)^{40} \approx 0.818$. Then readout on 10 qubits: $(0.97)^{10} \approx 0.737$. Combined: 0.60. - Cost estimate is biased — off by roughly 40% relative to true value.

With readout mitigation. - Readout bias corrected by tensor-product M matrix inversion. Residual readout error: $\sim 1\%$. Combined fidelity: $0.818 \cdot 0.99 \approx 0.81$. - Cost estimate bias: $\sim 19\%$.

Plus symmetry verification (particle number). - Discards shots that violate $\hat{N} = 5$. Roughly 15% of shots are discarded. - Effective shots: $0.85 \cdot 10^5 = 8.5 \times 10^4$. - Bias from remaining errors (those preserving particle number): $\sim 10\%$.

Plus ZNE (linear fit, $\lambda = 1, 1.5, 2$). - Uses 3x the shot budget ($3 \times 8.5 \times 10^4 = 2.55 \times 10^5$) to get the extrapolation points. - After linear fit, residual bias: $\sim 2\%$. - Variance roughly doubled compared to single $\lambda = 1$ run.

Final result: cost estimate with $\sim 2\%$ bias, requires $\sim 3\times$ shot count relative to raw. Acceptable for VQA optimization.

Observation. Each mitigation layer halves (or better) the residual bias. The stack is additive. Getting below 2% bias requires either CDR or PEC, both of which are significant investments.

Visualization

Pending. A bar chart showing bias and variance for each mitigation layer applied sequentially: (no mitigation) → +readout → +symmetry → +ZNE. Bias decreases; variance increases. Caption: "The bias-variance tradeoff of mitigation stacks."

Exercises

1. For a depolarizing channel at rate $p = 0.01$ and a single-qubit observable X, compute the PEC quasi-probability decomposition and the resulting γ.
2. Show that symmetry verification is *unbiased* — the estimator using only symmetric shots has the same expectation as the ideal estimator, provided the symmetry is exactly preserved by the ideal circuit.
3. (VQA) For a 20-qubit VQA with depth 100, gate fidelity 99.5%, readout fidelity 97%, and shot budget 10^6, design a mitigation stack and estimate the final cost bias and effective shot count.

Research Extensions

- **Adaptive mitigation** — selecting the mitigation technique based on the current circuit properties.
- **Learning-based mitigation** — neural networks trained to correct noisy expectation values given side information about the circuit.
- **Hybrid PEC/ZNE** — combining the two techniques to trade off bias and variance optimally.
- **Mitigation-aware optimizer design** — optimizers that account for the mitigation cost in their loss function.
- **Scalable virtual distillation** — reducing the qubit overhead of copy-based techniques.

Cross-links to Other Courses

- **Quantum Error Correction** — the long-term answer that mitigation approximates.
- **Statistical Estimation Theory** — bias-variance tradeoff as a fundamental constraint.
- **Module 9.5 (Zero-Noise Extrapolation)** — the detailed treatment of the most widely used technique.
- **Module 9.6 (Noise-Adaptive Ansatz Design)** — a structurally different mitigation strategy.
- **Module 10 (Adaptive Control Systems)** — mitigation-aware optimizer design.

Zero-Noise Extrapolation

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.5

Zero-noise extrapolation (ZNE) is the single most widely deployed error mitigation technique for VQAs. It is conceptually simple, needs no detailed noise model, works with any observable, and can be applied to almost any circuit with modest overhead. The basic idea:

1. Run the circuit at several **artificially amplified** noise levels.
2. Fit the expectation value as a function of noise strength.
3. Extrapolate back to zero noise.

The key insight is that you can deliberately make your circuits noisier in controlled amounts, and from the resulting data points you can infer where the line (or curve) would cross at zero noise. Under the right conditions, this gives a substantially better estimate than just running the circuit at nominal noise.

ZNE was introduced by Li-Benjamin (2017) and Temme-Bravyi-Gambetta (2017) more or less simultaneously. It has since become the default mitigation technique in Qiskit, Mitiq (a dedicated mitigation library), PennyLane, and most VQA benchmarking studies.

This node covers the details: noise amplification methods, fit choices, error analysis, practical implementation, and its integration with VQA optimization.

The Basic Idea, Formally

Let $\langle O \rangle(\lambda)$ be the expectation value of observable O when the circuit's noise is amplified by factor $\lambda \geq 1$. The nominal circuit corresponds to $\lambda = 1$. By construction, $\lambda = 0$ corresponds to *no noise*.

Assumption. The function $\lambda \mapsto \langle O \rangle(\lambda)$ is smooth and can be extrapolated from points $\lambda_1 = 1, \lambda_2, \lambda_3, \ldots$ back to $\lambda = 0$.

Procedure.

1. Choose a set of amplification levels $\{\lambda_i\}$ (e.g., $\{1, 1.5, 2, 2.5\}$).
2. For each λ_i, amplify the circuit's noise by that factor and run N_{shots} shots to get $\hat{O}(\lambda_i)$.
3. Fit a model $f(\lambda)$ to the data points $\{(\lambda_i, \hat{O}(\lambda_i))\}$.
4. Report the extrapolated value $\hat{O}_{\text{ZNE}} = f(0)$ as the mitigated estimate.

The critical choices are: **how to amplify the noise** (without changing the logic of the circuit) and **what functional form to fit**.

Noise Amplification Methods

Three main strategies exist for increasing the effective noise without changing the ideal operation of the circuit.

1. Unitary folding

Replace each gate U with $U \cdot U^\dagger \cdot U$, which is logically equivalent to U but has three times the gate count (and hence three times the noise).

More generally, folding by factor $\lambda = 2k + 1$ replaces U with $U \cdot (U^\dagger U)^k$. **Global folding** applies this to the entire circuit U_{circ}; **local folding** applies it to individual gates.

Pros: noise scaling is discrete ($\lambda = 1, 3, 5, \ldots$) but can be made continuous by folding a random subset of gates. Works in circuit-level simulators. Widely supported.

Cons: the assumption that noise scales linearly with gate count is only approximate (different gates have different noise, and noise doesn't always compose linearly).

Reference. Giurgica-Tiron et al. (2020) introduced the modern unitary folding framework.

2. Pulse stretching

At the pulse level, lengthen each gate's control pulses by a factor $\lambda \geq 1$. The gate does the same ideal operation but takes longer, during which more decoherence accumulates.

Pros: the noise amplification is *physical* rather than synthetic — you're actually running the hardware in a noisier regime. Fewer assumptions.

Cons: requires pulse-level hardware access (not available via simple circuit APIs), the stretching may not preserve gate fidelity exactly (coherent errors can get worse rather than just scaling), and λ is bounded by the coherence time (you can't stretch indefinitely).

Reference. Kandala et al. (2019) used pulse stretching for ZNE in the IBM hardware stack.

3. Identity insertion

Insert identity operations (physically realized as $UU^\dagger$ for some U) at various points in the circuit, adding noise without changing the ideal logic.

Similar to unitary folding in effect but more flexible about *where* the extra noise goes.

Pros: can target specific gates or qubits for amplification.

Cons: the specific choice of U matters (different U's have different noise profiles), so the amplification is not cleanly parameterized.

In practice. Unitary folding is the default for circuit-level ZNE; pulse stretching is the default when pulse-level access is available; identity insertion is used for specialized studies.

Extrapolation Fits

With data $\{(\lambda_i, \hat{O}_i)\}$, several fit choices exist.

Linear fit

$$f(\lambda) = a\lambda + b.$$

Extrapolated value: $f(0) = b$.

Simplest, robust, appropriate when the noise is small and the dependence is approximately linear.

Pros: stable, easy to implement, small variance.

Cons: can have significant bias when the true dependence is nonlinear.

Polynomial fit

$$f(\lambda) = \sum_{k=0}^{d} c_k \lambda^k.$$

Degree d fit; extrapolation is $f(0) = c_0$. Richardson extrapolation is a special case.

Pros: more flexible.

Cons: higher-degree polynomials overfit and amplify statistical noise in the extrapolation. Numerical instability for large d.

Typical choice: $d = 2$ or $d = 3$, no higher.

Exponential fit

$$f(\lambda) = Ae^{-\Gamma\lambda} + B.$$

Extrapolated value: $f(0) = A + B$.

The exponential form is motivated by the actual dependence of Pauli expectation values on depolarizing noise: $\langle P\rangle(\lambda) \approx \langle P\rangle_{\text{ideal}} \cdot (1 - 4p/3)^{L\lambda}$, which is exponential in λ.

Pros: matches the true depolarizing dependence.

Cons: the fit is nonlinear and can fail to converge with sparse data. Requires at least 3 data points and often 4+.

Typical choice: when $\geq$4 data points are available and noise is known to be approximately depolarizing.

Richardson extrapolation

A specific polynomial extrapolation that eliminates leading-order error terms one at a time. For example, with $\lambda = 1, 2$:

$$f(0) \approx 2\hat{O}(1) - \hat{O}(2).$$

With $\lambda = 1, 2, 3$: a weighted sum of the three values that cancels both linear and quadratic terms.

Pros: closed-form, known error bounds, well-studied in numerical analysis.

Cons: assumes specific polynomial structure, amplifies variance at higher orders.

Error Analysis

Two sources of error affect ZNE:

1. Statistical error. Each $\hat{O}(\lambda_i)$ has shot noise of order $1/\sqrt{N_{\text{shots}}}$. The extrapolation amplifies this noise. For linear extrapolation with two points $\lambda = 1, 2$, the extrapolated variance is:

$$\text{Var}[\hat{O}_{\text{ZNE}}] = \text{Var}[2\hat{O}(1) - \hat{O}(2)] = 4\sigma^2 + \sigma^2 = 5\sigma^2,$$

so the variance is 5x the single-run variance. For higher-order fits, the variance amplification is larger (up to $\sim 30\sigma^2$ for Richardson with $\lambda = 1, 2, 3$).

2. Extrapolation bias. If the true dependence is not the model you fit, extrapolation introduces a bias. For a linear fit applied to data that is actually exponential, the bias is $\sim$ (curvature) $\cdot\, \lambda_{\text{min}}^2$.

Tradeoff. Higher-order fits reduce bias but increase variance. The optimal choice depends on the circuit, noise level, and shot budget.

Rule of thumb. For moderate noise and shallow circuits, use linear fits. For moderate-to-deep circuits, use exponential. For high-precision work, use several fit types and compare — if they agree, trust them; if they disagree, the extrapolation is unreliable.

Pitfalls Of ZNE

Several situations where ZNE fails or misleads:

1. Non-monotonic dependence. If the noisy expectation value is not monotonic in λ (e.g., because of coherent errors that build up non-linearly), the extrapolation has no fixed point to converge to. Fit functions can give nonsense values.

2. Noise that's not a "factor". ZNE assumes noise can be amplified *uniformly*. For heterogeneous noise (different gates have very different error rates), this assumption breaks down. The effective λ is ambiguous.

3. Saturation at small λ. If the nominal noise is already close to saturation (noise level so high that expectation values are near zero), you cannot extrapolate linearly — the curve is in its flat region and the fit is ill-conditioned.

4. Cost landscape distortion. Applying ZNE to every cost evaluation in a VQA optimization slows convergence (because shots are 3-10x more expensive) *and* introduces correlated errors across iterations. The optimizer sees a slightly noisier and slightly warped landscape.

5. The expensive folded circuits. Folded circuits have more gates, so they take longer to run *and* they're more error-prone. At high λ, the amplified circuit may be so noisy that the data point is useless.

Mitigating the pitfalls. Use multiple λ values (at least 3), try multiple fit functions, check consistency, and validate on benchmark problems where the ideal answer is known (Clifford circuits work well for this).

ZNE In VQA Optimization

A subtle question: when should ZNE be applied *during* VQA optimization?

Option A — Every iteration. Apply ZNE at every cost evaluation. Guarantees the optimizer sees the mitigated cost, but the shot overhead is 3-10x per iteration.

Option B — Only at the final evaluation. Optimize with raw (unmitigated) cost, then apply ZNE once at the end to estimate the true cost at the converged point. Saves shots during training but relies on the raw cost having the same minimum as the mitigated cost.

Option C — Periodic mitigation. Apply ZNE every k iterations to check convergence, use raw cost otherwise. Compromise between A and B.

The tradeoff. Option B is correct *if* noise is depolarizing (since depolarizing preserves the minimum location — see node 9.2). For more structured noise, the raw minimum may differ from the mitigated minimum, and Option A is needed.

Current practice. Most 2026 VQA experiments use Option B or C for compute efficiency, with Option A reserved for small-scale precision studies.

ZNE As A Basic Building Block

ZNE is the "default" mitigation in most VQA libraries and is the first technique to reach for. Its main virtues:

1. **Model-free.** Doesn't require knowledge of the noise channel.
2. **Universal.** Works for any observable, any circuit, any noise type.
3. **Cheap.** Overhead is 3-10x in shots, which is manageable.
4. **Stackable.** Can be combined with readout mitigation, symmetry verification, DD without conflict.
5. **Transparent.** The extrapolation curve gives a visual diagnostic — if the curve looks wrong, you know ZNE is unreliable.

The main virtues of *alternatives* (CDR, PEC, virtual distillation) are corrections to specific ZNE failure modes. ZNE is the baseline; other techniques are improvements for specific conditions.

Structural Role

This node is the **detailed treatment** of the most widely used mitigation technique, expanding on the overview in node 9.4. It covers noise amplification methods, fit choices, error analysis, pitfalls, and VQA integration.

It is the reference node for ZNE in the course, and the practical companion to node 9.4's survey view.

Relations to Other Concepts

- **Li, Benjamin (2017)** — ZNE origin.
- **Temme, Bravyi, Gambetta (2017)** — ZNE + PEC origin.
- **Kandala et al. (2019)** — hardware ZNE demonstration with pulse stretching.
- **Giurgica-Tiron et al. (2020)** — unitary folding framework and Mitiq library.
- **LaRose et al. (2022)** — Mitiq: software package for error mitigation.
- **Richardson extrapolation (numerical analysis classic)** — the parent technique.
- **Module 9.2 (Depolarizing noise)** — why the minimum is preserved under depolarizing.
- **Module 9.4 (Mitigation techniques)** — the broader context.

Worked Example — ZNE On A 10-Qubit QAOA

Setup. A 10-qubit QAOA at depth $p = 3$, two-qubit gate fidelity 99.5%, shot budget 10^5 per cost evaluation. The cost is $\langle H_C \rangle = -0.53$ (noiseless, known from small simulation) and -0.32 (noisy, directly measured).

Step 1 — Choose amplification levels. $\lambda \in \{1, 2, 3\}$ via unitary folding.

Step 2 — Run circuits. - $\lambda = 1$: fold factor 1 (nominal), $\hat{O}_1 = -0.32$. - $\lambda = 2$: fold each gate to $UU^\dagger U$, 3 gate count, $\hat{O}_2 = -0.19$. - $\lambda = 3$: fold each gate twice, 5 gate count, $\hat{O}_3 = -0.11$.

Each shot budget 10^5, so total 3×10^5 shots.

Step 3 — Fit.

Linear fit: $f(\lambda) = -0.41 + 0.105\lambda$. $f(0) = -0.41$.

Exponential fit: $f(\lambda) = -0.55 \cdot e^{-0.42\lambda} + 0.01$. $f(0) = -0.54$.

Richardson (with $\lambda = 1, 2$): $f(0) = 2 \cdot (-0.32) - (-0.19) = -0.45$.

Step 4 — Compare to the truth. The noiseless value is -0.53. - Linear: -0.41, error 0.12. - Exponential: -0.54, error 0.01. **Best.** - Richardson: -0.45, error 0.08. - Unmitigated: -0.32, error 0.21.

Observation. Exponential is the best fit for this depolarizing-dominated regime. Linear is significantly biased because the underlying decay is not linear. Richardson is between the two.

Shot cost. 3x (three amplification levels $\times$ 10^5 shots each).

Variance amplification. Roughly 5x for linear, more for Richardson, less for exponential (because the fit is "softer").

Conclusion. Exponential ZNE gives a 20x bias reduction at 3x shot cost. Good tradeoff.

Visualization

Pending. A plot of $\langle H_C \rangle$ vs λ showing three data points (at $\lambda = 1, 2, 3$), three fit curves (linear, exponential, Richardson), and the true value at $\lambda = 0$. The exponential curve should hit the true value closely; the linear should miss. Caption: "ZNE in action: fit choice matters."

Exercises

1. For a single-qubit depolarizing channel at rate p, show that the expectation value of X under λ-folded noise is $(1 - 4p/3)^{2\lambda-1} \cdot \langle X \rangle_{\text{ideal}}$. Use this to derive the exact exponential fit formula.
2. Prove that linear ZNE is unbiased if the true $\langle O \rangle(\lambda)$ is linear in λ, and biased by $\mathcal{O}(\lambda^2 \cdot |O''|)$ if it is quadratic.
3. (VQA) For a 12-qubit VQE with depth 60 and two-qubit gate fidelity 99.3%, compare linear, exponential, and Richardson ZNE using simulated data. Which gives the smallest total error (bias + statistical)?

Research Extensions

- **Adaptive λ-selection** — automatically choosing the amplification levels based on the observed curvature.
- **Learning-based extrapolation** — using neural networks to extrapolate rather than parametric fits.
- **Joint cost/gradient ZNE** — mitigating both the cost and its gradient simultaneously with a shared shot budget.
- **Exponential ZNE with variance reduction** — variance reduction techniques specific to the exponential fit.
- **ZNE in the deep-circuit regime** — adapting ZNE for circuits that are past the saturation depth.

Cross-links to Other Courses

- **Numerical Analysis** — Richardson extrapolation, polynomial interpolation.
- **Statistical Estimation** — bias-variance tradeoff, extrapolation risk.
- **Module 9.4 (Mitigation techniques)** — overview context.
- **Module 9.8 (Taxonomy of Noise)** — where ZNE fits in the mitigation landscape.
- **Module 10.5 (Closing the Loop)** — integrating mitigation with the optimizer.

Noise-Adaptive Ansatz Design

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.6

Noise mitigation (node 9.4-9.5) treats noise as a fact to correct *after* the circuit runs. **Noise-adaptive ansatz design** takes the opposite view: modify the ansatz *before* the circuit runs, so that the noise profile of the specific hardware has the least harmful effect.

The motivating observation: noise on real hardware is *heterogeneous*. Some qubits are better than others. Some two-qubit gates (on specific qubit pairs) have much higher fidelity than others. The hardware connectivity graph has topology (not every pair of qubits can interact directly). And the noise spectrum changes slowly over time with drift and recalibration.

A noise-blind ansatz ignores all this and treats the circuit as a uniform object. A noise-adaptive ansatz uses the hardware information to:

1. **Place the best-fidelity qubits where the circuit needs them most.**
2. **Route two-qubit gates through the lowest-noise pairs.**
3. **Reduce depth in regions where coherence is weakest.**
4. **Choose the ansatz shape itself** to match the hardware's native gate set.

The result: better circuit fidelity for the same logical operation, at zero quantum runtime cost (all the optimization is classical). This is essentially a form of *hardware-aware compilation* specialized for VQAs.

This node covers the main techniques: qubit selection, routing, Hamiltonian-to-hardware mapping, hardware-efficient ansätze, and the ADAPT-VQE family of noise-aware ansatz growth.

Qubit Selection

The simplest form of noise-adaptive design. Modern superconducting hardware has 50-1000 qubits; most VQA problems use far fewer. The question: which subset of qubits do you pick?

Naive approach: use qubits 0 through $n-1$ on the device. Simple but ignores fidelity differences.

Better approach: rank qubits by fidelity metrics and pick the best n.

Useful metrics for ranking:

- **T1, T2 coherence times** — higher is better.
- **Readout fidelity** — higher is better.
- **Single-qubit gate fidelity** — higher is better.
- **Two-qubit gate fidelity on connecting edges** — higher is better.
- **Calibration freshness** — more recently calibrated qubits are usually better.

The ranking is done per-VQA-run using the daily calibration data published by the hardware provider (IBM, Rigetti, IonQ, etc.). The best qubits change over time as the hardware drifts.

Gotcha: the "best" qubits individually may not have the best two-qubit couplings. You need to optimize over *subgraphs*, not individual qubits. This is a graph optimization problem and can be hard in general (subgraph isomorphism is NP-hard), but practical heuristics work well for small problem sizes.

Reference. Tannu-Qureshi (2019) showed 3-5x fidelity improvement on IBM hardware via hardware-aware qubit selection alone.

Gate Routing

A VQA circuit usually implies a *logical* connectivity pattern (which qubits interact in the cost Hamiltonian). The hardware has a *physical* connectivity graph (which qubits are directly coupled). These rarely match exactly.

The **routing problem** is: map logical qubits to physical qubits, inserting SWAP gates where needed, to realize the logical circuit on the physical hardware with minimum noise cost.

Naive routing minimizes the total SWAP count — fewer SWAPs means fewer noisy operations.

Noise-aware routing weights each potential SWAP by the error rate of the specific two-qubit gate used. A high-fidelity SWAP pathway is preferred even if it's longer.

Formalization. This is a weighted graph optimization problem. Given the logical connectivity graph G_L and the physical graph G_P with edge weights w_{ij} (error rates), find a mapping and SWAP sequence that minimizes the total weighted error.

Practical tools. Qiskit's `SabreSwap`, `SabreLayout`, and `NoiseAdaptiveLayout` passes do noise-aware routing. Similar tools exist in Cirq (Google) and tket (Quantinuum).

Impact. On hardware with significant fidelity heterogeneity, noise-aware routing gives 2-5x fidelity improvement compared to naive routing.

Hardware-Efficient Ansätze (HEA)

A deeper form of adaptation: choose the ansatz *shape* to match the hardware's native gate set and connectivity.

Standard HEA:

$$U(\boldsymbol{\theta}) = \prod_{l=1}^{L} \left[\left(\prod_i R_Y(\theta_i^l) R_Z(\phi_i^l) \right) \cdot \left(\prod_{(i,j)\in E} \mathrm{CNOT}_{ij} \right) \right],$$

where E is the hardware's native two-qubit connectivity. Key properties:

- **Uses only native gates** — no expensive decomposition of arbitrary U into single-qubit rotations and CNOTs.
- **Respects connectivity** — no SWAP insertion required, since two-qubit gates are only between connected qubits.
- **Linearly parameterized in layers** — depth is the main knob.

Tradeoff. HEA has the best hardware fidelity *but* the worst expressibility per parameter. Hardware-native gates often don't span the full unitary group cleanly, so you need more layers to reach the same expressibility as a problem-tailored ansatz.

Practical observation. For NISQ VQAs, the fidelity savings often outweigh the expressibility loss. HEA with $L = 5 - 20$ layers is competitive with much deeper "theoretically better" ansätze on noisy hardware.

Reference. Kandala et al. (2017) — the original hardware-efficient VQE on IBM hardware for small molecules.

Hamiltonian-to-Hardware Matching

A more sophisticated technique: match the *structure* of the Hamiltonian to the hardware topology.

Observation. A chemistry Hamiltonian has locality structure — some terms involve adjacent orbitals, others involve distant ones. Mapping this to the hardware:

1. **Place orbitals that interact strongly on well-connected physical qubits.**
2. **Use SWAP networks that match the locality pattern.**
3. **Schedule the simulation to minimize communication between distant qubits.**

For Hamiltonians derived from lattice models (Hubbard, Heisenberg, Ising), this is even cleaner: the lattice structure directly maps to a hardware connectivity pattern, and the best ansatz is one that mirrors that lattice.

For chemistry VQEs specifically, the **unitary coupled cluster (UCC)** ansatz has a natural locality pattern inherited from Slater-Condon rules. Recent work (Lee et al. 2021) shows that matching UCC excitation operators to hardware topology improves fidelity 2-10x for 10-20 qubit molecules.

For QAOA on graph problems, a good rule is to *assign graph vertices to physical qubits such that adjacent vertices are adjacent on the hardware*. This reduces the SWAP count dramatically on native problems.

ADAPT-VQE And Growing Ansätze

The **Adaptive Derivative-Assembled Pseudo-Trotter (ADAPT-VQE)** method grows the ansatz one gate at a time, selecting at each step the gate that gives the largest gradient component.

Procedure.

1. Start with an empty ansatz (just the reference state).
2. Compute the gradient of the cost with respect to adding each candidate gate from a pre-defined pool.
3. Add the gate with the largest gradient magnitude.
4. Optimize the parameters of the current ansatz.
5. Repeat until convergence.

Why this is noise-adaptive. The gradient estimate is based on the noisy circuit, so the *noise-favored* gates tend to be selected — gates that are high-fidelity on the specific hardware give larger and more reliable gradients, so they enter the ansatz first. The result is an ansatz that is implicitly hardware-aware.

Pros: extremely compact ansätze (much fewer parameters than HEA), naturally hardware-aware, grows only as much as needed for the specific problem.

Cons: the gradient computation at each growth step is expensive (you're computing the gradient over the full operator pool). For large pools, this becomes a significant overhead.

Reference. Grimsley, Economou, Barnes, Mayhall (2019) introduced ADAPT-VQE. Subsequent work extended it to fermionic-qubit mappings and noise-aware variants.

Gate Fidelity As A Cost Term

A direct approach to noise-adaptive design: add the gate-noise cost directly to the training objective.

Idea. Define a modified cost:

$$C_{\text{adapted}} = C_{\text{quantum}} + \alpha \cdot C_{\text{noise}},$$

where $C_{\text{noise}} = \sum_g p_g$ is the sum of noise rates over all gates in the circuit, and α is a regularization parameter.

Optimization now trades off the quantum cost against the noise cost: the optimizer is encouraged to find low-gate-count, low-error solutions.

Caveat. The gate count is discrete (number of gates in the circuit), so gradient descent doesn't directly apply. You need either: - A soft penalty (α times a smooth approximation of gate count). - Discrete search

(genetic algorithms, reinforcement learning, Bayesian optimization). - A structured ansatz family where gate count is controlled by a continuous parameter.

Practical use. Primarily a research direction — not yet standard. The thesis's HOPSO framework can naturally incorporate a noise-aware cost term in its objective generator, which is a potential future extension.

Dynamic Ansatz Reconfiguration

A futuristic version: the ansatz structure is **reconfigured during training** based on observed hardware fidelity changes.

Procedure.

1. Monitor the hardware fidelity in real time (via periodic benchmarking).
2. Detect when specific gates have degraded (e.g., due to drift or a recalibration).
3. Re-route or re-layer the ansatz to avoid the degraded gates.
4. Continue optimization with the new ansatz structure.

This is a form of **adaptive circuit compilation during training** and fits naturally into the control-theoretic view of Module 7. It's the logical extension of node 7.6's feedback control framework to the ansatz level.

Research status. Early experiments (Ziman et al. 2023, Hetzinger et al. 2024) show this can improve final fidelity by 2-5x on drifting hardware. Not yet standard; active research.

Noise-Adaptive Vs Mitigation: Structural Comparison

Dimension	Mitigation (9.4, 9.5)	Adaptation (9.6)
When applied	After runs	Before runs
Cost domain	Shot count	Classical compile time
What it fixes	Residual bias	Systematic fidelity
Depth limit	Extends by 3-10x	Extends by 2-5x
Combines with	Other mitigation	Mitigation techniques
Hardware info needed	Minimal	Full calibration data
Best for	Moderate noise	Heterogeneous noise

The two approaches are complementary. You should apply *both*: a noise-adaptive ansatz to minimize the underlying error, plus mitigation to further reduce residual bias. The combined improvement is roughly multiplicative.

The Thesis View

Noise-adaptive design fits the thesis framework in a clean way:

1. **It is a structural modification of the objective generator.** The objective function C is unchanged conceptually, but its hardware representation changes.
2. **HOPSO can incorporate hardware data as a parameter.** The optimizer can be given the current fidelity map as a static input, and it can be designed to prefer parameter settings that activate high-fidelity gates more than low-fidelity ones.

3. **Separation of dynamics and objective.** The *dynamics* of HOPSO (how particles move) is unchanged; only the *objective landscape* (what they're searching) is adapted. This is the right way to think about it architecturally.

4. **Adaptive regime.** The static noise-adaptive design of section 9.6 extends naturally to the dynamic one of Module 10 — the objective generator can be updated during training as hardware drifts, closing the loop between hardware monitoring and circuit structure.

Structural Role

This node is the **ansatz-level mitigation** complement to nodes 9.4-9.5 (post-processing mitigation). It covers qubit selection, gate routing, hardware-efficient ansätze, Hamiltonian-to-hardware matching, ADAPT-VQE, and dynamic reconfiguration.

It is the structural precondition for Module 10's adaptive VQA architectures, which take the noise-adaptive ideas and embed them in a feedback-control framework.

Relations to Other Concepts

- **Tannu, Qureshi (2019)** — hardware-aware qubit selection.
- **Kandala et al. (2017)** — original hardware-efficient ansatz.
- **Grimsley et al. (2019)** — ADAPT-VQE.
- **Li, Benjamin, Proctor (2020)** — noise-aware compilation with SabreSwap-style algorithms.
- **Module 9.4 (Mitigation)** — complementary post-processing approach.
- **Module 7.6 (Feedback control)** — dynamic reconfiguration as closed-loop control.
- **Module 10 (Adaptive VQA)** — the deployment endpoint.

Worked Example — 8-Qubit Chemistry VQE On A Heterogeneous Device

Setup. An 8-qubit H_2O chemistry VQE on a 27-qubit IBM device (call it `ibm_kyoto` for concreteness). Hardware calibration data: - Single-qubit gate fidelities: 99.95%-99.99%. - Two-qubit gate fidelities: 98.5%-99.6% (varies by edge). - Best 8-qubit connected subgraph: qubits $\{3, 5, 8, 11, 14, 16, 19, 22\}$ with average two-qubit fidelity 99.4%. - Worst 8-qubit connected subgraph: $\{0, 1, 2, 4, 6, 7, 9, 10\}$ with average two-qubit fidelity 98.8%.

Circuit. A UCCSD ansatz with 50 two-qubit gates. Run with 10^5 shots per cost evaluation.

Comparison:

Strategy	Qubit subset	Avg 2q fid	Circuit fid	Cost bias
Naive (qubits 0-7)	worst	0.988	0.55	45%
Noise-aware selection	best	0.994	0.74	26%
Best + noise-aware routing	best	0.994 + routing	0.78	22%
Best + routing + HEA switch	best	—	0.83	17%
Best + routing + HEA + ZNE	best	—	—	6%
Best + routing + HEA + ZNE + CDR	best	—	—	2%

Observation. Noise-adaptive design alone reduces bias by ~30% (45% → 17%). Adding mitigation on top reduces it further (17% → 2%). The stack multiplies: the total improvement is ~20x.

Compute cost. Qubit selection and routing: seconds. HEA conversion: minutes. Training: same as baseline. ZNE: 3x shots. CDR: 1.5x more compute for Clifford regression. Net: ~5x compute cost total for 20x bias reduction. Worth it.

Visualization

Pending. A diagram of a 27-qubit device with two 8-qubit subsets highlighted: the naive one (bottom-left, mostly-worst) and the noise-aware one (best-fidelity subset). Edges are colored by two-qubit gate fidelity. Caption: "Qubit selection: picking the right 8 out of 27 matters."

Exercises

1. Given a hardware graph $G_P = (V, E, w)$ with edge weights w_{ij} (error rates) and a desired subset size n, formulate the noise-aware qubit selection problem as a graph optimization problem. Is it tractable?
2. For a QAOA on a 10-vertex ring-graph MaxCut, design the noise-aware qubit-to-vertex mapping on a linear-connectivity hardware device. Compare SWAP count to naive mapping.
3. (VQA) Implement a noise-aware variant of the parameter-shift gradient that weights each gate's gradient by its fidelity. Does this improve convergence on a noisy 6-qubit VQA?

Research Extensions

- **Adaptive noise-aware compilation** — recompile during training as hardware drifts.
- **Noise-aware parameter initialization** — start near parameter settings that favor high-fidelity gates.
- **Multi-objective ansatz search** — jointly optimize expressivity and hardware fidelity.
- **Hardware-co-designed ansätze** — design the ansatz *and* the hardware gate set together for specific target problems.
- **Transfer learning for ansatz structure** — reuse ansatz designs across similar problems on similar hardware.

Cross-links to Other Courses

- **Compilers and code generation** — routing and layout as compilation problems.
- **Graph theory** — subgraph selection and minimum-weight routing.
- **Machine learning architecture search** — ADAPT-VQE as a form of NAS (neural architecture search) for quantum circuits.
- **Module 9.4 (Mitigation)** — the post-processing complement.
- **Module 10 (Adaptive Control Systems)** — the closed-loop ansatz adaptation.

Simulation Modes: Statevector, Density Matrix, Aer

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.7

Before you run a VQA on real hardware, you almost always simulate it classically. Simulation serves several purposes: developing and debugging the algorithm, evaluating expressivity and trainability, testing noise mitigation, and producing baselines against which to compare the hardware results. Getting simulation right is foundational to the whole VQA research enterprise.

But there isn't one kind of quantum simulation — there are several, each with different capabilities, costs, and use cases. The main modes:

1. **Statevector simulation** — exact, noiseless, stores the full 2^n-dimensional amplitude vector. Fastest for small n, exponentially expensive.
2. **Density matrix simulation** — exact, supports mixed states and general noise channels, stores a $2^n \times 2^n$ matrix. Doubles the memory cost compared to statevector.
3. **Stochastic / trajectory simulation (Qiskit Aer, quantum trajectory methods)** — Monte Carlo sampling of pure-state trajectories under noise. Memory like statevector but with a shot overhead for noise averaging.
4. **Tensor network simulation (MPS, PEPS)** — exact or approximate, uses low-entanglement structure. Scales better for certain classes of circuits.
5. **Clifford simulation (stabilizer formalism)** — exact and efficient for Clifford-only circuits. Limited to Clifford gates + Pauli observables.
6. **Approximate classical simulation** — methods that produce an approximation of the state with a budget.

This node covers the main modes relevant to VQAs: statevector (the fastest for small n), density matrix (the most expressive noise model), Aer-style stochastic trajectories (the practical middle ground), and tensor networks (the scaling solution for structured states). Understanding the tradeoffs is essential for designing VQA experiments that can actually be simulated within a reasonable compute budget.

Statevector Simulation

The simplest mode: store the amplitude vector $|\psi\rangle \in \mathbb{C}^{2^n}$ directly, and apply each gate as a matrix-vector multiplication.

Memory. $2^n \times 16$ bytes (complex doubles) $= 2^{n+4}$ bytes. For $n = 30$, that's 16 GB (feasible on a workstation). For $n = 35$, 512 GB (big machine). For $n = 40$, 16 TB (supercomputer). The effective limit is around $n = 30 - 35$ on commodity hardware.

Compute. Each gate is an $O(2^n)$ operation (for single-qubit gates) or $O(2^n)$ for two-qubit gates (with careful implementation). A depth-L circuit costs $O(L \cdot 2^n)$.

Noise handling. Statevector simulation is *noiseless*. If you want to simulate noise, you need to either switch to a different mode (density matrix, trajectory) or inject the noise as classical randomness (e.g., stochastically applying Pauli errors between gates).

Pros: fastest for small n, simplest code, no approximation.

Cons: exponential in n, no native noise support.

Use cases for VQAs: - Developing and debugging the algorithm structure. - Computing exact expectation values to serve as ground truth for noisier simulations. - Evaluating expressibility (via frame potential sampling) and trainability (via exact gradient variance). - Small-scale proof-of-concept experiments.

Qiskit's `AerSimulator(method='statevector')`, Cirq's `Simulator`, PennyLane's `default.qubit`, and Intel QS are typical implementations.

Density Matrix Simulation

Stores the full density matrix $\rho \in \mathbb{C}^{2^n \times 2^n}$. This is the most general state representation — it can handle mixed states, which are required for anything involving noise, partial traces, or decoherence.

Memory. $2^{2n} \times 16$ bytes. For $n = 15$, 16 GB. For $n = 18$, 1 TB. Effective limit is $n \approx 15$ on a workstation.

Compute. Each gate application is a superoperator action on ρ, costing $O(4^n)$ naively but can be made $O(2^n \cdot k)$ for k-local gates with smart implementation. Noise channels are applied the same way as gates, using Kraus operators.

Pros: full generality, exact noise simulation with any Kraus channel, clean handling of partial traces and reduced states.

Cons: memory cost is squared compared to statevector — half the qubit limit ($n \approx 15$ vs 30).

Use cases for VQAs: - Testing noise mitigation techniques on small systems. - Comparing different noise models (depolarizing vs amplitude damping). - Computing exact noisy gradients and their variances. - Validating noise-aware ansatz designs.

When to use. When you need to simulate noise and your system is small enough to afford the memory cost.

Qiskit's `AerSimulator(method='density_matrix')` and `qiskit_aer.noise.NoiseModel` is the standard implementation.

Stochastic Trajectory Simulation

The practical middle ground: use pure-state (statevector) simulation, but randomly inject noise at each gate according to the noise channel's Kraus decomposition.

How it works. For each gate followed by a noise channel with Kraus operators $\{K_k\}$, sample a k with probability $\|K_k|\psi\rangle\|^2$ and apply $K_k|\psi\rangle / \|K_k|\psi\rangle\|$. Repeat many times and average.

Memory. Same as statevector: $O(2^n)$. Can simulate up to $n \approx 30$.

Compute. Same as statevector per trajectory, but you need many trajectories to average out the noise. Typically 100-10000 trajectories per data point.

Pros: scales to large n, can handle any Kraus channel, no exponential blowup from density matrix.

Cons: each trajectory is noisy — the simulated observable estimate has variance from *both* the original shot noise *and* the Monte Carlo noise averaging. Can require many more trajectories than naive shot count.

Use cases for VQAs: - Simulating noisy VQAs on 20-30 qubits with realistic noise. - Testing mitigation techniques at scale. - Producing the baseline "noisy simulation" against which hardware runs are compared.

When to use. When you need noisy simulation *and* density matrix is too expensive (typically $n > 15$).

Qiskit Aer's `AerSimulator(method='matrix_product_state')` and `method='stabilizer'` are not trajectory methods, but the default `method='automatic'` often uses trajectory methods when noise is present. PennyLane's `default.mixed` and Cirq's `density_matrix_simulator` are alternatives.

Tensor Network Simulation (MPS, PEPS)

For quantum states with *low entanglement*, tensor network methods scale much better than the general-state methods.

Matrix Product States (MPS). Write the state as

$$|\psi\rangle = \sum_{i_1,\ldots,i_n} \mathrm{Tr}(A_1^{i_1} A_2^{i_2} \cdots A_n^{i_n}) |i_1 \cdots i_n\rangle,$$

where $A_k^{i_k}$ are matrices of dimension $\chi \times \chi$ (called the "bond dimension").

Memory. $O(n \cdot \chi^2)$. Grows polynomially in n and in χ.

Compute. Each two-qubit gate costs $O(n \cdot \chi^3)$ in the worst case.

Key observation. The bond dimension χ is related to the *entanglement* in the state: it must be at least $2^{S/\ln 2}$ where S is the entanglement entropy between a subsystem. For *low-entanglement states*, χ stays small and simulation is efficient.

Pros: can simulate very large systems (hundreds of qubits) for low-entanglement states. Handles noise via purification (adding ancilla legs).

Cons: fails for highly entangled states — the bond dimension explodes. Also awkward for non-local interactions (two-qubit gates between distant qubits).

Use cases for VQAs: - Benchmarking VQAs on large systems (50-200 qubits) when the state remains low-entanglement. - Testing circuits on lattice Hamiltonians where the ansatz preserves locality. - Dequantization studies — seeing if the quantum state could be simulated classically.

When *not* to use. For highly entangled circuits (random HEA, deep QAOA), MPS fails. Statevector or trajectory is the better choice.

Reference. Vidal (2004) introduced time-evolving block decimation (TEBD); modern packages include ITensor, quimb, and TeNPy.

2D generalization (PEPS). Projected Entangled Pair States work on 2D lattices. Useful for 2D VQAs but have their own convergence issues.

Clifford Simulation

A very different mode: the **stabilizer formalism** tracks quantum states as collections of "stabilizers" (Pauli operators that the state is an eigenstate of) rather than amplitudes.

Gottesman-Knill theorem. Any quantum circuit consisting of: - Clifford gates (CNOT, H, S, their conjugates) - Pauli measurements - Preparation in computational basis states

can be simulated classically in polynomial time — specifically, $O(n^3)$ per gate for n qubits.

Pros: exponentially faster than statevector for Clifford-only circuits. Can simulate thousands of qubits.

Cons: *only* for Clifford circuits. Any non-Clifford gate (T, parameterized rotations) breaks the formalism. Can be extended to "T-doped" circuits with $O(2^t)$ cost for t T gates.

Use cases for VQAs: - Testing the Clifford variants of a VQA (e.g., a VQA reduced to Clifford for benchmarking). - Clifford Data Regression (node 9.4) uses this as a subroutine. - Noise characterization via randomized benchmarking (Clifford circuits with varied lengths).

When to use. Whenever you can reduce your VQA to a Clifford subcircuit, use Clifford simulation for that part.

Qiskit Aer's `method='stabilizer'`, Cirq's Clifford simulator, and Aaronson's CHP are standard implementations.

Approximate Classical Simulation

Several methods produce *approximate* classical simulations of quantum states at controlled accuracy:

1. Truncated amplitude methods. Only track the largest amplitudes, discard small ones. Works when the state is sparse in the computational basis.

2. Classical shadows (Huang-Kueng-Preskill 2020). Measure the state in random bases and use the measurement record to compute arbitrary k-local observables. Not a full simulation but sufficient for VQA expectation values.

3. Slicing methods. Break the circuit into smaller slices that are simulated independently and then combined. Scales with circuit depth but with cost reduction from parallelism.

4. Machine-learning surrogates. Train a neural network to approximate the expectation value function $\boldsymbol{\theta} \mapsto C(\boldsymbol{\theta})$ from a small number of exact evaluations. Useful for exploring the landscape without running the simulator.

Use cases. These are for extreme scaling or exploration. Not primary simulation modes but complementary tools.

Simulation Mode Decision Tree

For a VQA practitioner, the choice of simulation mode depends on:

1. **System size n.**
   - $n \leq 15$: density matrix is affordable, use it for noise simulation. Statevector for noiseless.
   - $15 < n \leq 30$: statevector for noiseless, stochastic trajectory for noisy.
   - $n > 30$: tensor network if low entanglement, classical shadows for expectation values, or restrict to Clifford variants.
2. **Noise model needed?**
   - No noise: statevector, tensor network (if large).
   - Depolarizing only: stochastic trajectory is cheap.
   - General noise model: density matrix (if small enough), trajectory (if large).
3. **What's being computed?**
   - Full state: density matrix or statevector.
   - Expectation values only: classical shadows can work.
   - Gradient: parameter shift rule with any underlying simulator.
4. **Expressibility/trainability study?**
   - Noiseless statevector — no noise needed for the structural analysis.
5. **Mitigation validation?**
   - Density matrix for small, trajectory for medium, unavailable for very large.

Rule of thumb. Start with the cheapest mode that answers your question, and only upgrade when the cheaper mode is insufficient. Noise simulation is expensive; avoid it when possible.

Thesis View: Simulation Modes And The Objective Generator

From the thesis framework:

1. **The simulation mode *is* the objective generator for classical pretraining.** Before deploying to hardware, the VQA's cost function is computed via simulation; the simulation mode determines the accuracy and speed of that computation.

2. **Different simulation modes give different signal-to-noise tradeoffs.** Exact (statevector) simulation gives perfect signal; trajectory simulation matches hardware noise. Choosing the mode is a design decision that affects the training dynamics.
3. **HOPSO can use different simulators for different purposes.** Exact for expressibility checks, trajectory for realistic cost evaluation, classical shadow for large-scale exploration. The separation of objective and dynamics lets you swap simulators without changing the optimizer.
4. **Simulation is the "oracle" for the optimizer.** The optimizer doesn't know whether its cost evaluations come from hardware or simulation — as long as the interface is the same, the optimizer works. This is the practical manifestation of the thesis's abstraction.

Structural Role

This node is the **practical simulation reference** for VQA research. It covers the main simulation modes, their costs, and when to use each.

It is the bridge between the theoretical noise models (9.1-9.3) and the practical experiments that use them. Module 9's material is only useful if you can actually simulate it, and this node is the guide to doing so.

Relations to Other Concepts

- **Gottesman, Knill (1998)** — stabilizer formalism and Clifford simulation.
- **Vidal (2004)** — TEBD and MPS for quantum simulation.
- **Aaronson, Gottesman (2004)** — improved Clifford simulation.
- **Huang, Kueng, Preskill (2020)** — classical shadows.
- **Qiskit Aer documentation** — the reference implementation.
- **Module 9.1 (Quantum noise channels)** — what the simulator needs to represent.
- **Module 9.4 (Mitigation)** — what the simulator is used to test.

Worked Example — Choosing A Simulator For A 20-Qubit VQE

Problem. A 20-qubit chemistry VQE on LiH with a hardware-efficient ansatz at depth 10. Goal: evaluate the exact noiseless cost and a noisy cost with 99.5% two-qubit gate fidelity, 99.9% single-qubit fidelity, 97% readout fidelity.

Option 1 — Statevector, noiseless. - Memory: $2^{20} \cdot 16$ bytes = 16 MB. Trivial. - Compute: $10 \cdot 20 \cdot 2^{20} \approx 2 \times 10^8$ operations per data point. Fast (seconds). - Use: for the noiseless baseline.

Option 2 — Density matrix, noisy. - Memory: $2^{40} \cdot 16$ bytes = 16 TB. Unfeasible on workstation. - Skip this option.

Option 3 — Stochastic trajectory, noisy. - Memory: $2^{20} \cdot 16$ bytes = 16 MB per trajectory. - Compute: same as statevector per trajectory. Need ~1000 trajectories per data point. - Total compute: 1000 $\times$ (statevector cost) $\approx 2 \times 10^{11}$ operations $\approx$ minutes per data point. - Use: for the noisy baseline.

Option 4 — Hardware. - Compute: 1-10 minutes per cost evaluation including queue time. - Noise level: realistic. - Cost: real compute time and shot budget.

Workflow. Use Option 1 for landscape exploration and hyperparameter tuning; Option 3 for validating noise mitigation; Option 4 for the final runs.

Observation. For 20-qubit VQEs, density matrix simulation is already out of reach. Trajectory methods become essential. This is typical for realistic VQA experiments.

Visualization

Pending. A plot comparing simulation cost (memory, time) vs qubit count for each mode: statevector, density matrix, trajectory, MPS (at fixed bond dimension), Clifford. Log-log scale. Caption: "Simulation modes scale differently. Pick the right one."

Exercises

1. Compute the memory cost of density matrix simulation for $n = 10, 15, 20, 25$ qubits. At what point does it exceed your workstation's RAM?
2. For stochastic trajectory simulation, show that the variance of the noisy expectation value estimate is proportional to $1/N_{\text{traj}}$ and derive the constant.
3. (VQA) For a 25-qubit MPS simulation of a depth-20 HEA, estimate the bond dimension needed for 1% accuracy. Is this tractable?

Research Extensions

- **Hybrid simulation modes** — switching between simulators during a VQA run based on resource availability.
- **GPU-accelerated statevector simulation** — scaling statevector beyond $n = 30$ using multi-GPU clusters.
- **Smart noise injection** — reducing the trajectory count needed by stratified sampling.
- **Approximate density matrix methods** — low-rank or tensorized density matrices for $n > 15$.
- **Machine-learning surrogates for cost landscapes** — neural networks replacing expensive simulation during optimization.

Cross-links to Other Courses

- **Classical Computing and Algorithms** — cost models, memory hierarchy, parallelization.
- **Tensor Network Methods in Physics** — MPS, PEPS, DMRG.
- **Module 9.1 (Quantum noise channels)** — what simulators represent.
- **Module 9.4 (Mitigation techniques)** — what simulators are used to validate.
- **Module 8.7 (Quantum advantage criteria)** — the line between tractable and intractable simulation.

Taxonomy of Noise in VQA

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.8

This is the **closing synthesis** of Module 9. The earlier nodes covered noise as individual objects: quantum channels (9.1), the depolarizing model (9.2), real hardware parameters (9.3), mitigation techniques (9.4-9.5), noise-adaptive ansatz design (9.6), and simulation modes (9.7).

This node steps back and organizes the whole landscape into a **taxonomy**: a structured map of the kinds of noise that affect VQAs, the kinds of damage they do, and the kinds of defenses that apply to each. The aim is to give a single chart that a practitioner can consult when diagnosing a noisy VQA — *"what kind of noise is killing me, and what do I do about it?"*

The taxonomy is organized along three axes:

1. **Where in the stack the noise lives** (physical, gate, circuit, classical post-processing).
2. **What kind of effect it has on the quantum state** (unital vs non-unital, Markovian vs non-Markovian, coherent vs incoherent).
3. **What it does to the VQA cost landscape** (shrinks, shifts, flattens, warps).

Each cell in the taxonomy has a characteristic signature, a set of mitigation techniques, and a connection to the thesis framing. At the end, I close the module with a summary of where the VQA literature stands on noise (2026) and where the open problems are.

Axis 1: Where The Noise Lives

Quantum computing has a layered architecture, and noise can enter at any layer. Knowing the layer tells you the characteristic timescale and the available countermeasures.

Layer 1 — Physical (sub-gate)

Examples. Thermal excitation, spontaneous emission, flux noise in superconducting qubits, laser phase noise for ion traps, atom loss for neutral atoms, nuclear spin coupling for silicon spins.

Timescale. Continuous during the circuit. Parameterized by T_1, T_2, T_ϕ (node 9.3).

Effect. Coherent and incoherent decay of the quantum state. The dominant source of decoherence in all current platforms.

Countermeasures. Hardware engineering (better qubits, better isolation, dynamical decoupling). Cannot be fixed at the circuit level; it's intrinsic to the physical layer.

Layer 2 — Gate (control-level)

Examples. Calibration drift, imperfect pulse shape, crosstalk between neighboring gates, leakage outside the computational subspace, finite control precision.

Timescale. Per-gate (nanoseconds to microseconds for superconducting, microseconds for ions).

Effect. Gate fidelity less than 1. Causes coherent errors (when control parameters are slightly wrong) and incoherent errors (when control parameters vary randomly).

Countermeasures. Recalibration (periodic benchmarking and correction), pulse shaping (DRAG pulses, Slepian pulses), randomized compiling (converting coherent to incoherent), gate-level Pauli twirling.

Layer 3 — Circuit (compilation-level)

Examples. SWAP overhead from routing, suboptimal layout, unnecessary gate count from poor decomposition, idle-time errors during synchronization.

Timescale. Circuit-level (microseconds to milliseconds).

Effect. Inflates the effective noise rate compared to the optimal circuit. Essentially a "compilation inefficiency" that translates to extra noise.

Countermeasures. Better compilation: noise-aware routing, native-gate ansätze, circuit optimization passes, ADAPT-style ansatz growth.

Layer 4 — Measurement (readout)

Examples. Readout confusion (measuring 0 when true state is 1), dispersive shift errors, thermal population.

Timescale. Measurement time (microseconds).

Effect. Biases the measurement outcome. Typically 1-5% on current hardware — larger than gate errors.

Countermeasures. Readout error correction (confusion matrix inversion, M3), repeated measurement, ancilla-based parity checks.

Layer 5 — Classical post-processing

Examples. Numerical precision in the classical optimizer, shot noise in expectation value estimation, optimizer instability, hyperparameter sensitivity.

Timescale. Classical loop iteration (milliseconds to seconds).

Effect. Adds noise to the effective cost function. Not a quantum noise per se, but functionally similar from the optimizer's perspective.

Countermeasures. More shots, better estimator (Bayesian posteriors, filters), optimizer improvements (Adam, adaptive learning rates), regularization.

Axis 2: What Kind Of Effect On The State

Within the quantum channel formalism, noise channels can be classified by several structural properties.

Unital vs Non-Unital

A channel is **unital** if $\mathcal{E}(I) = I$ — it preserves the maximally mixed state.

- **Unital examples:** depolarizing, Pauli noise, dephasing.
- **Non-unital examples:** amplitude damping (has a preferred state $|0\rangle$), thermal relaxation, measurement disturbance.

Practical significance. Unital noise has no "preferred direction" in state space — it symmetrically shrinks expectation values. Non-unital noise *biases* the state toward a specific fixed point, which means it can *move the minimum* of the VQA cost landscape, not just shrink its amplitude. This is a more insidious kind of error, because the optimizer converges to the wrong answer.

Markovian vs Non-Markovian

A noise process is **Markovian** if the error at each step depends only on the current state (no memory). It is **non-Markovian** if there's memory — the error depends on the history.

- **Markovian examples:** depolarizing channels (applied per gate), random Pauli noise.

- **Non-Markovian examples:** low-frequency flux noise, 1/f noise, crosstalk between distant gates via a shared bath.

Practical significance. Markovian noise accumulates predictably (multiplicatively per gate). Non-Markovian noise is harder — it can cause correlated errors that break standard analysis. Most VQA theory assumes Markovian; real hardware is often not.

Coherent vs Incoherent

Coherent errors are unitary but wrong — e.g., a Z rotation by 1.01π instead of π. They preserve the purity of the state; the state evolves to a slightly wrong pure state.

Incoherent errors are mixing operations — they increase the state's entropy. Decoherence, depolarizing noise, amplitude damping.

Practical significance. Coherent errors are *more damaging per error rate* because they can constructively interfere over many gates, compounding in a correlated way. Incoherent errors average out (variance scales as $\sqrt{L}$). So a 1% coherent error can do more damage than a 1% incoherent error over a deep circuit.

Countermeasure. *Randomized compiling* converts coherent errors to incoherent by randomly conjugating each gate with a random Pauli. It's essentially free and recommended for VQAs on coherent-error-dominated hardware.

Axis 3: What It Does To The VQA Cost Landscape

This is the VQA-specific axis — what the noise does to the optimizer's experience of the landscape.

Shrinks The Contrast

The most common effect: depolarizing-like noise shrinks the amplitude of all Pauli expectation values uniformly. The cost landscape is compressed toward a constant, but its *shape* (location of minima, directions of gradients) is unchanged.

Signature. Cost values all closer to the noiseless average $\mathrm{Tr}(H)/d$; minimum unchanged in parameter space; gradients scaled down.

Impact on VQA. Reduces signal-to-noise ratio; optimizer still finds the right answer but needs more shots. Mitigable via ZNE or CDR.

Shifts The Minimum

Non-unital noise (amplitude damping, thermal population) biases the state toward a specific fixed point, which shifts the cost function's minimum in parameter space. The optimizer converges, but to the wrong place.

Signature. Noisy minimum is at different parameters than noiseless minimum; even with infinite shots, the optimizer converges to a biased answer.

Impact on VQA. Systematic bias that cannot be removed by averaging. Requires either PEC (unbiased mitigation) or symmetry verification (if a symmetry is broken by the noise).

Flattens The Landscape

Depolarizing noise at high rate or over deep circuits causes the entire landscape to become flat — the exponential decay of contrast reaches the shot noise floor everywhere. This is the **noise-induced barren plateau** (Wang 2021).

Signature. Gradient variance is indistinguishable from shot noise; optimization stalls; no parameter update direction is reliable.

Impact on VQA. Training fails. No amount of shots helps (shot noise floor is the problem). Only defense is shallower circuits or noise reduction at the hardware level.

Warps The Landscape

Coherent errors and correlated noise can introduce *wrong* features into the landscape — spurious minima, new saddle points, distorted gradient directions.

Signature. Hard to detect; the optimizer converges but the convergence point is wrong in a non-obvious way. Validation against a known answer reveals the issue.

Impact on VQA. Insidious. Testing on a known-answer benchmark (small H_2 or a Clifford-reducible problem) is essential to catch this.

The Full Taxonomy Table

Putting it all together:

Noise type	Layer	Unital?	Markov?	Cost effect	Mitigation
Depolarizing	Gate	Yes	Yes	Shrinks	ZNE, CDR
Amplitude damping	Physical/Gate	**No**	~Yes	Shifts + shrinks	PEC, symmetry verification
Dephasing	Physical	Yes	Yes (approx)	Shrinks Z contrast	DD, ZNE
Coherent (cal drift)	Gate	Yes	Yes	Warps	Recalibration, randomized compiling
Crosstalk	Gate	Varies	**No**	Warps	Scheduling, DD, isolation
Leakage	Gate	**No**	Varies	Warps	Leakage-reducing pulses, filtering
Readout	Measurement	N/A	Yes	Bias observations	Confusion matrix inversion
1/f flux noise	Physical	Yes	**No**	Shrinks + warps	DD, hardware engineering
Shot noise	Classical	N/A	Yes	Flattens	More shots, Bayesian filter
Calibration drift	Control	Varies	**No**	Warps over time	Periodic recalibration, adaptive control

Reading the table. Each row is a distinct noise source. The columns tell you how it behaves and what to do about it. The "Mitigation" column points at specific techniques.

Key observations: - Non-unital and non-Markovian noise are the hardest cases. - Depolarizing is the best-case scenario — everything else is worse. - Classical-layer "noise" (shot noise, drift) is a major factor in practice.

The VQA Noise Hierarchy

On current (2026) hardware, the noise sources rank roughly as follows, in order of total impact on VQAs:

1. **Two-qubit gate errors.** By far the dominant source. 0.1%-1% per gate. All mitigation efforts focus here first.
2. **Readout errors.** Second-biggest. 1%-5% per qubit. Cheap to fix, always apply readout mitigation.
3. **Single-qubit gate errors.** 0.01%-0.1%. Smaller but accumulate over many gates.
4. **Decoherence during idle.** T2 effects. Mitigable via DD and smart scheduling.
5. **Crosstalk.** Usually smaller than direct gate errors but can be correlated and insidious.
6. **Coherent errors / calibration drift.** Slow-varying. Recalibration handles it.
7. **Leakage.** Usually $< 0.1\%$ on mature platforms. Reduced by careful pulse design.

This hierarchy is *specific to superconducting hardware in 2026*. Trapped ions have different rankings (less readout error, more laser noise). Neutral atoms add atom loss. The taxonomy is platform-dependent but the axes are universal.

Where The Field Stands (2026)

A summary of the state of noise + VQA research as of 2026:

1. Theory is mature for depolarizing noise. Wang 2021's plateau theorem, Ragone-style bounds, and ZNE theory give a comprehensive picture for the depolarizing case.

2. Non-unital noise is less well-understood. Analytical results for amplitude damping are scattered. Most theoretical work still uses depolarizing as a stand-in.

3. Mitigation is well-developed but has hit a shot-complexity wall. ZNE, PEC, CDR, symmetry verification are all established. The open question is how to extend them without exponential shot blow-up.

4. Hardware fidelities are improving but slowly. Two-qubit gate fidelities have gone from 99% (2018) to 99.9% (2026) on the best platforms — a factor of 10. This gives a factor-10 increase in usable depth. Steady but not transformational progress.

5. Circuit-level mitigation is well-established. Noise-aware compilation, ADAPT-VQE, and HEA are in every production VQA stack.

6. Open frontiers: - **Non-Markovian noise modeling** — how to treat it in VQA theory. - **Adaptive mitigation** — auto-selecting techniques based on circuit properties. - **Noise-aware optimizer design** — optimizers that know about mitigation cost. - **Error-corrected VQAs** — using minimal error correction (a few logical qubits) inside VQAs. - **Noise-tailored ansätze** — designing ansätze that are specifically resistant to the dominant noise types.

The field has matured from "we don't know how noise affects VQAs" (2017) to "we have a comprehensive taxonomy and standard tools" (2026). The remaining questions are about *how much* mitigation and adaptation can push the NISQ-era depth, and when fault-tolerance will take over.

Thesis Framing

From the thesis perspective:

1. **Noise is a perturbation of the objective generator, not of the optimizer dynamics.** The optimizer (HOPSO or otherwise) operates on whatever noisy cost function it's given; the noise lives in the evaluation of the objective. Separating these cleanly is architecturally important.
2. **Mitigation is a classical side-computation attached to the objective generator.** Not part of the quantum dynamics. This matches HOPSO's abstraction of the optimizer-objective boundary.

3. **Noise-adaptive ansatz design is a modification of the objective generator's functional form.** Changing the ansatz changes what function $C(\boldsymbol{\theta})$ the optimizer sees. This is a form of dynamical restriction that complements regularization.

4. **Adaptive noise handling (monitoring hardware drift and reconfiguring) is the natural extension.** This is what Module 10 takes up explicitly: the closed-loop adaptive VQA that uses feedback control at the outer level.

The thesis claim from this module: **noise is a solvable problem at the depths current hardware allows, provided you use the full toolkit (adaptation + mitigation + hardware tuning) and choose the right optimizer for the noisy landscape**.

Closing Module 9

Module 9 has traversed the noise landscape:

- **9.1:** Quantum channels and CPTP maps — the formal language.
- **9.2:** Depolarizing noise — the workhorse model and its implications.
- **9.3:** Gate errors and coherence times — the physical parameters.
- **9.4:** Mitigation techniques — the post-processing toolkit.
- **9.5:** Zero-noise extrapolation — the dominant mitigation in depth.
- **9.6:** Noise-adaptive ansatz design — the pre-processing complement.
- **9.7:** Simulation modes — how to model all this classically.
- **9.8 (this node):** Taxonomy — the full chart and the state of the field.

The arc: from abstract formalism → to physical reality → to defensive techniques → to a unified map. Each step adds structure that the previous lacked.

This module is the *practical* companion to Modules 7 (control theory) and 10 (adaptive deployment). Those modules tell you how to *design* a VQA for noise; this module tells you what the noise *is*.

Structural Role

This node is the **synthesis and closing** of Module 9. It organizes the module's content into a three-axis taxonomy, provides a summary table, ranks noise sources by impact, and sketches the state of the field in 2026.

It is the capstone reference node for VQA practitioners who need to quickly diagnose noise issues and select countermeasures.

Relations to Other Concepts

- **Nielsen-Chuang** — the formal channel language.
- **Wang et al. (2021)** — noise-induced plateau theorem.
- **Cai et al. (2023)** — comprehensive review of quantum error mitigation.
- **Module 9.1-9.7** — the components being synthesized.
- **Module 10** — the next step, where adaptive control handles drift and time-varying noise.

Worked Example — Diagnosing A Failing VQA

Scenario. You're running a 12-qubit VQE for a molecule. You know the noiseless answer (from exact diagonalization): ground state energy $E_0 = -7.88$ Hartree. Your noisy VQA converges to -7.72, off by 0.16 Hartree (way too much).

Diagnosis workflow:

Step 1. Apply readout error correction. New result: -7.79. Improvement 0.07; readout was a significant contributor.

Step 2. Apply symmetry verification on electron count. New result: -7.81. Improvement 0.02; non-unital (amplitude-damping-like) errors were shifting the minimum.

Step 3. Apply ZNE with linear fit. New result: -7.85. Improvement 0.04; depolarizing-like errors were shrinking contrast.

Step 4. Switch to noise-aware qubit selection (best 12 qubits on the device). New result: -7.87. Improvement 0.02; heterogeneous fidelity was inflating effective noise.

Step 5. Apply DD during idle periods. New result: -7.875. Improvement 0.005.

Remaining error: 0.005 Hartree. Possibly unmitigable or requires deeper techniques (PEC, CDR).

Diagnosis summary. - Main source: readout error (44% of total bias). - Second: depolarizing contrast loss (25%). - Third: non-unital shift (12%). - Fourth: heterogeneous fidelity (12%). - Fifth: idle errors (3%).

Each source is attacked by a different mitigation. The taxonomy directs which tool to use for which problem.

Visualization

Pending. A treemap or Sankey diagram showing the five-layer stack (physical, gate, circuit, measurement, classical), with noise sources in each layer sized by their typical contribution to VQA error. Arrows connecting each source to its mitigation technique. Caption: "The noise taxonomy at a glance."

Exercises

1. For each noise type in the taxonomy table, identify whether it can shift the VQA minimum (non-unital) or just shrink contrast (unital).
2. (Conceptual) Why is a 1% coherent error typically more damaging than a 1% incoherent error for a deep VQA? Quantify using a circuit of depth 100.
3. (VQA, project-scale) Design a noise diagnostic pipeline: given a noisy VQA result, determine the dominant error source by applying mitigation layers one at a time and measuring the improvement. Output the diagnosis as a table.

Research Extensions

- **Non-Markovian noise for VQAs** — extending the theory to time-correlated errors.
- **Adaptive mitigation selection** — auto-picking the right mitigation per circuit.
- **Error-correction-aware VQAs** — running VQAs on partially error-corrected circuits.
- **Noise-tailored ansatz families** — ansätze specifically designed to resist the dominant error type on the target hardware.
- **Unified VQA noise simulator** — simulating all noise axes simultaneously with controllable knobs.

Cross-links to Other Courses

- **Module 9.1-9.7** — all the components of the taxonomy.
- **Module 7 (Control-Theoretic View)** — the design side.
- **Module 10 (Adaptive Control Systems)** — the deployment endpoint.
- **Module 6 (Optimization Dynamics)** — the optimizer's interaction with noise.
- **Quantum Error Correction** — the long-term answer beyond NISQ.

Module 10 — Vqa As Adaptive Control Systems

Adaptive VQA Architectures

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.1

Module 10 is the **deployment chapter** of the thesis. The earlier modules built up structural (1-4), dynamical (5-6), control-theoretic (7), expressivity-theoretic (8), and hardware-noise (9) views. This module answers: **what does a VQA actually look like when you take all of that seriously and build it for real hardware?**

The answer, in one phrase: **an adaptive control system**. The central claim is that VQAs are not just machine-learning-style optimization problems — they are closed-loop control systems running on noisy, drifting, heterogeneous hardware, and their architecture should reflect that. Module 7.6 planted this thesis; this module develops it into specific architectural patterns and deployment strategies.

This opening node, 10.1, defines what an **adaptive VQA architecture** is and lays out the building blocks. Subsequent nodes explore specific components (measurement-adaptive circuits in 10.2, RL-based VQAs in 10.3, quantum control applications in 10.4, closed-loop systems in 10.5, the future in 10.6, HOPSO-as-controller in 10.7, measurement-optimizer co-design in 10.8).

What Makes A VQA "Adaptive"?

A non-adaptive (static) VQA:

- Has a fixed ansatz.
- Has a fixed optimizer with fixed hyperparameters.
- Evaluates the cost using a fixed shot budget.
- Runs for a fixed number of iterations or until a fixed convergence criterion.
- Ignores the hardware's state (fidelity, drift, queue depth).

An adaptive VQA:

- Modifies the ansatz based on observed performance (adaptive ansatz growth, ADAPT-VQE).
- Modifies the optimizer hyperparameters based on observed gradient statistics (learning rate scheduling, trust regions).
- Modifies the shot budget based on cost uncertainty (shot-adaptive estimation).
- Modifies the iteration schedule based on convergence state (early stopping, restart-on-stuck).
- Monitors the hardware (fidelity drift, recalibration, queue latency) and adjusts accordingly.

Each of these "modifications" is a feedback signal. The system observes its own state, decides what to do, and acts. This is the feedback control pattern from Module 7.6, specialized to the VQA case.

Building Blocks Of An Adaptive VQA

A full adaptive VQA has several distinct components. Think of each as a subsystem with its own purpose.

1. Ansatz controller

Role. Decides the ansatz shape: which gates to include, in what order, on which qubits, with what connectivity.

Inputs. Current parameter values, cost history, gradient statistics, hardware fidelity map.

Actions. Grow the ansatz (add a gate), prune (remove an ineffective gate), reconfigure (change qubit mapping), reshape (switch to a different ansatz family).

Example. ADAPT-VQE grows the ansatz by adding the highest-gradient operator at each step. A drift-aware variant also tracks hardware fidelity and avoids growing into degraded regions.

2. Optimizer controller

Role. Runs the parameter optimization itself: gradient descent, Adam, HOPSO, etc.

Inputs. Current cost and gradient, historical trajectory, variance estimates.

Actions. Update parameters, adjust learning rate, restart, switch to a different optimizer.

Example. A gradient descent with a learning-rate controller that decreases the step size when gradient variance is high (shot noise dominated) and increases it when variance is low (signal dominated).

3. Estimator controller

Role. Manages the cost/gradient estimation procedure: how many shots, what mitigation to apply, which circuits to run.

Inputs. Target precision, available shot budget, noise model.

Actions. Allocate shots across circuits, select mitigation techniques, compute filtered estimates.

Example. A shot allocator that gives more shots to terms with large coefficients in the Hamiltonian, less to negligible terms.

4. Hardware monitor

Role. Tracks the state of the hardware: calibration freshness, gate fidelities, queue depth, errors during execution.

Inputs. Hardware calibration reports (usually hourly-daily), execution metadata, mid-circuit errors.

Actions. Trigger recalibration requests, change hardware target, alert if degradation is severe.

Example. A monitor that detects when the best qubits have drifted below a fidelity threshold and reroutes the circuit to a different subset.

5. Meta-controller

Role. Coordinates the above. Decides when each controller should act and in what order.

Inputs. State of all subsystems, user goals, resource budgets.

Actions. Trigger ansatz growth, switch optimizers, shut down and restart, report status.

Example. A top-level loop that runs the optimizer for 20 iterations, then grows the ansatz if cost has plateaued, then re-optimizes, repeating until the total compute budget is exhausted.

The Two-Loop Structure Revisited

Module 7.1 introduced the **two-loop structure** of a VQA: inner loop (quantum, microseconds) and outer loop (classical, seconds). The adaptive VQA adds a *third* loop on top:

Inner (quantum). Gate-level circuit execution, microseconds per gate. Unitary evolution plus physical noise.

Outer (classical optimizer). Parameter update, seconds per iteration. Gradient descent or Bayesian optimization or HOPSO. Observes the cost, computes a direction, takes a step.

Meta (adaptation). Architecture reconfiguration, minutes to hours. Monitors the outer loop's performance, decides when to grow/prune the ansatz, when to switch optimizer hyperparameters, when to rerun hardware calibration.

This **three-loop structure** is the natural generalization. Each loop runs at its own timescale, with the outer loop nested inside the meta loop and the inner loop nested inside the outer.

Comparison to classical control: - Inner = physical plant (the qubit dynamics). - Outer = feedback controller (the optimizer). - Meta = adaptive supervisor (the architecture manager).

This maps directly to the **hierarchical control** pattern in classical systems engineering: fast inner controllers handle the physics, slower outer controllers handle the dynamics, and slow meta-controllers handle the architecture.

Timescale Separation

A key principle: the three loops must operate at **well-separated timescales** to avoid destructive feedback.

- Inner: nanoseconds (gate-level).
- Outer: seconds (parameter update).
- Meta: minutes+ (architecture change).

If timescales overlap, the loops interfere: for example, if you change the ansatz mid-optimization, the outer loop's gradient estimates become meaningless (the landscape moved), so the optimizer can't converge. Similarly, if you recalibrate hardware mid-circuit, the inner and outer loops are out of sync.

Design principle. Each loop should run to quasi-convergence before the next-higher loop acts. The outer loop should reach a stable minimum before the meta loop changes the ansatz. The inner loop should finish execution before the outer loop observes the result.

This is standard practice in classical hierarchical control and should be standard in VQA design.

Adaptive Ansatz Patterns

Specific adaptive architectural patterns:

Growing ansätze

Start small, grow one piece at a time based on observed gradients. ADAPT-VQE is the canonical example. Variants:

- **ADAPT-VQE** (Grimsley 2019): add the operator with largest gradient.
- **Iterative qubit coupled cluster (iQCC)** (Ryabinkin 2020): growth via UCC-style operators.
- **Layer-wise learning** (Skolik 2021): add one HEA layer at a time, freezing previous layers.
- **Reinforcement-learning ansatz search** (module 10.3): RL agent decides the next gate.

Advantage: compact ansätze, hardware-aware growth, natural stopping criteria.

Pruning ansätze

Start large, remove gates that aren't contributing. Analogous to neural network pruning.

- **PARTYVQE** and similar: track gate gradient magnitudes and remove the smallest.
- **Sparse ansatz rewriting**: detect and eliminate redundant gate sequences.

Advantage: can start from a standard HEA and simplify; benefits VQAs that over-parameterize.

Mutation patterns

Genetic algorithms or evolutionary strategies that modify the ansatz via random mutations (add/remove/swap gates) and select for improved cost.

Advantage: global search over ansatz space, not stuck in local architectural minima.

Disadvantage: slow, expensive in compute, often impractical for large problems.

Structured reconfiguration

Ansatz family is fixed (e.g., HEA) but parameters like depth, qubit assignment, or connectivity graph are adapted based on performance.

Advantage: simpler than full ansatz search, benefits from HEA's hardware-native structure.

Adaptive Optimizer Patterns

Specific optimizer adaptations:

Learning rate scheduling

Adjust the learning rate based on observed cost history: - Decay on plateau (decrease lr when cost stops improving). - Cosine annealing (smooth decrease over a schedule). - Warmup (start small, increase, then decrease).

Gradient variance adaptation

Adjust the learning rate *or* shot count based on gradient variance: - High variance $\rightarrow$ lower lr or more shots. - Low variance $\rightarrow$ higher lr.

Trust region methods

Limit each parameter step to a trusted region around the current point. Expand or shrink the region based on how well predicted cost decrease matches observed.

Optimizer switching

Run gradient descent for exploration, switch to Adam for fine-tuning, switch to L-BFGS for final convergence. Each optimizer has a regime where it excels.

HOPSO-style population tracking

Maintain a population of particles with diverse parameters, adapt the population based on fitness variance, blend gradient-based and swarm-based updates. HOPSO is the thesis's flagship example.

Adaptive Estimator Patterns

Shot budget allocation

For a Hamiltonian decomposed as $H = \sum_i c_i P_i$, estimate each $\langle P_i \rangle$ with an allocation of shots proportional to $|c_i|$. Wasted shots on negligible terms are avoided.

Mitigation selection

Apply readout mitigation always. Apply ZNE if depth is moderate. Apply CDR if Clifford simulation is feasible. Skip expensive mitigation on easy circuits.

Early stopping

If the cost has converged within some tolerance, stop taking shots and return the estimate. Avoids wasting shots on low-value updates.

Bayesian filtering

Maintain a posterior over the cost value and update it as shots come in. Stop when the posterior is concentrated enough for the optimizer's needs.

The Thesis Claim

The thesis's specific claim in this module: **HOPSO, combined with an adaptive architecture, is the right control system for NISQ-era VQAs**.

The argument:

1. HOPSO separates the dynamical update from the objective generator. This separation is the right architectural boundary for adaptive control.
2. HOPSO's hybrid gradient+population structure naturally handles both exploration (population) and exploitation (gradient), which an adaptive controller needs.
3. HOPSO's tunable dynamics (via regularization) give the meta-controller levers to adjust optimizer behavior without reimplementing the optimizer.
4. An adaptive VQA using HOPSO as its optimizer controller inherits this clean architecture and benefits from the thesis-level analysis of when HOPSO converges.

This is the module's structural argument. Subsequent nodes develop specific pieces: RL-based adaptation (10.3), HOPSO-as-controller (10.7), and measurement-optimizer co-design (10.8).

Structural Role

This node is the **opening survey** of Module 10. It defines what adaptive VQA means, lays out the building blocks (ansatz, optimizer, estimator, hardware monitor, meta-controller), and establishes the three-loop hierarchy that the rest of the module develops.

It is the entry point for the module. All other nodes build on this framework.

Relations to Other Concepts

- **Module 7.6 (Feedback control)** — the parent framework.
- **Module 9.6 (Noise-adaptive ansatz)** — the hardware-awareness component.
- **ADAPT-VQE (Grimsley 2019)** — the canonical growing-ansatz example.
- **Åström, Wittenmark (1995)** — *Adaptive Control*, the classical reference.
- **Sutton, Barto (2018)** — *Reinforcement Learning*, the parent of RL-based VQAs.
- **Module 10.7** — the thesis's HOPSO-as-controller specialization.

Worked Example — A Full Adaptive VQE Architecture

Problem. 20-qubit chemistry VQE for a medium molecule (C_4H_8) on noisy hardware.

Architecture:

Inner loop (quantum). Hardware-efficient ansatz with initial depth $L = 5$. Execution time per circuit: 100 μs. Repeated for 1000 shots per cost evaluation.

Outer loop (optimizer). HOPSO with 10 particles. Starts with learning rate 0.1. Runs for 100 iterations or until cost plateau.

Estimator. Readout mitigation always on. ZNE with linear fit at $\lambda \in \{1, 2\}$ every 5 iterations. Shot allocation weighted by Hamiltonian term coefficients.

Meta loop (architecture controller). - Every 20 iterations: check gradient variance. If variance drops below threshold, grow ansatz by adding 1 layer. If variance high, apply DD. - Every 100 iterations: check hardware calibration. If best-qubit fidelity has dropped by 20%, re-route the circuit. - Overall: run up to 500 iterations total, increasing ansatz depth by 5 layers max. Stop early if cost improvement per iteration $<$ 0.001 for 50 iterations.

Observations.

1. The inner loop is unchanged from a standard VQA.
2. The outer loop uses HOPSO for natural exploration/exploitation balance.
3. The meta loop makes the architecture responsive to observed dynamics.
4. Each timescale is well-separated: inner 100 μs, outer seconds, meta minutes.
5. Each component is modular — can be swapped independently.

This is not a toy — it's the kind of architecture 2026 research-grade VQA experiments actually use. The challenge is *engineering* the meta-controller to make good decisions, not conceptualizing the architecture.

Visualization

Pending. A three-loop diagram: inner (quantum execution), outer (optimizer), meta (architecture). Each loop has its own inputs, actions, and timescale. Arrows show the information flow between loops. Caption: "Adaptive VQA: three nested loops at three timescales."

Exercises

1. For a VQA with inner loop 100 μs, outer loop 1 s, meta loop 1 min, verify the timescale separation is at least 3 orders of magnitude per loop. Why does this matter for stability?
2. Design a meta-controller that decides when to grow the ansatz based on: (a) cost convergence, (b) gradient variance, (c) hardware calibration freshness. Specify the logic.
3. (VQA) Implement a minimal adaptive VQA for a 4-qubit QAOA with HOPSO as the optimizer and a simple meta-controller that grows the depth every 50 iterations. Compare convergence to a fixed-depth baseline.

Research Extensions

- **Stability analysis of three-loop VQAs** — when do the loops interfere destructively?
- **RL-based meta-controllers** — learn the meta-controller's policy from data.
- **Multi-device adaptive VQAs** — the meta-controller manages execution across multiple hardware backends.

- **Online architecture search** — search the ansatz space during training, not as a separate preprocessing step.
- **Formal guarantees for adaptive VQAs** — extending Robbins-Monro-style convergence to the meta-loop.

Cross-links to Other Courses

- **Adaptive Control (Åström-Wittenmark)** — the classical theory.
- **Reinforcement Learning (Sutton-Barto)** — the RL meta-controller pattern.
- **Hierarchical Control Systems** — the timescale-separation principle.
- **Module 7.6 (Feedback Control)** — the precursor.
- **Module 9.6 (Noise-adaptive ansatz)** — the hardware-awareness component.

Measurement-Adaptive Circuits

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.2

A **measurement-adaptive circuit** is one where the gates applied *after* a certain point depend on the outcome of a measurement taken *during* the circuit execution. Instead of running a fixed sequence of gates, the circuit branches based on measurement outcomes and different branches apply different gates.

This is a substantial departure from the standard "prepare, evolve, measure" structure of quantum circuits. In the standard model, measurements happen only at the end. In the adaptive model, measurements can happen mid-circuit and influence subsequent gates. This is sometimes called **mid-circuit measurement** or **dynamic circuit execution**.

Measurement-adaptive circuits are the quantum analogue of **conditional execution** in classical computing — the `if` statement of quantum programming. They unlock several capabilities that static circuits cannot match:

1. **Quantum error correction** — measure stabilizers, decode syndromes, apply correction operations. Fundamentally requires mid-circuit measurement.
2. **Measurement-based quantum computation (MBQC)** — the whole computation is a sequence of adaptive measurements.
3. **Gate teleportation** — move a gate from one qubit to another using an ancilla and a classically-conditioned correction.
4. **Adaptive state preparation** — prepare a target state more reliably by measuring and conditionally correcting.
5. **Noise-adaptive VQAs** — the VQA version: measure the environment mid-circuit and adapt the rest of the circuit accordingly.

The hardware requirements are strict: fast (sub-microsecond) classical feedback, high-fidelity measurement, and the ability to apply gates conditional on classical bits. Most superconducting platforms have added these capabilities in 2023-2025. Trapped ions have had them for longer.

This node covers the principles, the VQA applications, and the emerging research on **measurement-adaptive VQAs**.

The Hardware Requirement

For a measurement-adaptive circuit to work in practice, the hardware must support:

1. **Mid-circuit measurement.** You must be able to measure a qubit partway through a circuit without destroying the rest of the computation.
2. **Fast classical feedback.** The measurement result must reach the classical controller and a decision must be made fast enough to influence the next gates — before the remaining qubits have decohered.
3. **Conditional gates.** The hardware must support applying a gate conditional on a classical bit.

2026 hardware status:

- **Superconducting (IBM, Google, Rigetti):** all three capabilities are available with 100 ns - 1 μs feedback latency. This is fast enough to matter for circuits with longer coherence time.
- **Trapped ions (IonQ, Quantinuum):** feedback latency 100 μs - 1 ms (slower). Usable for deep circuits since ion coherence times are very long.
- **Neutral atoms (QuEra):** slower still, 1-10 ms. Usable for specific MBQC-style architectures.
- **Silicon spin qubits:** limited mid-circuit measurement availability.

The platform choice significantly affects what's practical. IBM and Google's superconducting platforms in 2026 are the best fit for most VQA applications of measurement-adaptive circuits.

The Formal Structure

A measurement-adaptive circuit has a structure that's not easily captured by the standard "circuit = sequence of unitaries" picture. Instead, think of it as a *program*:

```
prepare initial state |ψ_0⟩
apply U_1
apply U_2(θ_1)
measure qubit q_1, result = m_1
if m_1 = 0:
    apply U_3(θ_2)
else:
    apply U_4(θ_3)
apply U_5(θ_4)
measure qubit q_2, result = m_2
if m_2 = 0:
    apply U_6(θ_5)
else:
    apply U_7(θ_6)
measure remaining qubits
```

The program tree is exponentially large in the worst case (each branch point doubles the number of paths), but in practice the branches are shallow and well-structured.

Formally, a measurement-adaptive circuit is a **quantum channel** applied to the input state, where the channel has Kraus operators parameterized by measurement outcomes:

$$\mathcal{E}(\rho) = \sum_{m_1, m_2, \ldots} p(m_1, m_2, \ldots) \cdot U(\boldsymbol{\theta}; m_1, m_2, \ldots) \rho U^\dagger(\boldsymbol{\theta}; m_1, m_2, \ldots),$$

where each $U(\boldsymbol{\theta}; m)$ is the unitary applied along the branch corresponding to outcome trajectory m, and $p(m)$ is the probability of that trajectory.

Why VQAs Want Adaptive Circuits

Several compelling reasons:

1. More expressive with less depth. An adaptive circuit can implement states that a fixed circuit of the same depth cannot. The classical bits from mid-circuit measurement carry information that would otherwise require more gates.

Concrete example: preparing a GHZ state. A non-adaptive circuit needs $O(n)$ depth. An adaptive circuit can do it in $O(\log n)$ depth using parallel preparation and classical correction.

2. Better noise tolerance. If a mid-circuit measurement detects an error, you can apply a correction before the error propagates through the rest of the circuit. This is the error-correction paradigm applied inside a variational algorithm.

3. Native support for symmetry enforcement. Measure the conserved quantity mid-circuit, reject or correct if it's wrong, continue. More efficient than post-processing-only symmetry verification.

4. Access to non-unitary dynamics. Open-system dynamics, dissipative state preparation, measurement-based cooling — all require adaptive measurements. VQAs that target non-unitary targets can use adaptive circuits directly.

5. Entanglement at a distance. Gate teleportation via Bell pair distribution lets you apply two-qubit gates between arbitrary qubits (not just adjacent ones), relaxing the hardware connectivity constraint.

Adaptive Measurement Inside VQA Optimization

A natural VQA design question: *where* should the adaptive measurements go, and *how* should the controller (classical feedback) decide what to do?

Case 1 — Pre-programmed branches. The branches are fixed (part of the ansatz); only the parameter values are variational. The VQA's parameters $\boldsymbol{\theta}$ include parameters used in each branch. The cost function averages over measurement outcomes.

Cost function:

$$C(\boldsymbol{\theta}) = \sum_m p(m|\boldsymbol{\theta}) \cdot \langle H \rangle_{\text{branch } m}.$$

Case 2 — Learned branches. The branches are themselves parameterized by a classical decision function that's learned alongside the ansatz parameters. The decision function takes the measurement outcome and produces a gate choice.

Case 3 — Fully adaptive (MBQC-style). Every gate in the circuit is a measurement, and the parameters $\boldsymbol{\theta}$ are the measurement angles. The computation is defined by the graph of entanglement + measurement sequence.

VQA literature as of 2026 uses mostly Case 1 (fixed branches), with Cases 2 and 3 as emerging research directions.

Gradients Through Adaptive Circuits

An important technical issue: **how do you compute gradients through a measurement-adaptive circuit?**

The parameter-shift rule works for unitaries, not for measurements. When the circuit has mid-circuit measurements, the standard parameter-shift rule needs modification.

Approach 1 — Branch-wise parameter shift. For each branch, compute the gradient using parameter shift on the unitaries within that branch. Sum the gradients weighted by the branch probabilities.

Approach 2 — Parameter-shift for measurement angles. For MBQC-style circuits with measurement angle parameters, a specialized parameter-shift rule exists (the measurement angle is analogous to a rotation angle).

Approach 3 — Finite differences. Numerically differentiate by running the circuit at $\theta + \epsilon$ and $\theta - \epsilon$. Works but higher variance than parameter-shift.

Approach 4 — Policy gradient (for learned decision functions). Treat the classical decision function as a policy and use REINFORCE-style policy gradient estimators. Introduces additional variance but handles discrete decisions naturally.

Most VQA libraries (Qiskit, PennyLane) support Approach 1 for simple adaptive circuits as of 2026. Approach 4 is a research-stage technique.

Example: GHZ Preparation With Adaptive Circuits

A classic example of adaptive circuit speedup.

Goal. Prepare the n-qubit GHZ state $|\text{GHZ}\rangle = (|0\cdots0\rangle + |1\cdots1\rangle)/\sqrt{2}$.

Non-adaptive circuit. Starting from $|0\cdots0\rangle$: apply H to qubit 0, then CNOT(0,1), CNOT(1,2), ..., CNOT(n-2, n-1). Depth: n.

Adaptive circuit. Starting from $|0\cdots0\rangle$: 1. Apply H to all n qubits. 2. Measure all qubits in the Bell basis (pairwise CNOT + Hadamard + Z measurement). 3. Based on measurement outcomes, apply corrections to turn the post-measurement state into GHZ.

The depth is $O(\log n)$ (from the measurement circuit) plus classical feedback. For $n = 100$, this is a factor-10 reduction.

Practical caveat. The adaptive version requires a feedback loop with fast classical decisions, which introduces its own latency. For short coherence time hardware, the wall-clock advantage may be smaller than the gate-count argument suggests.

Adaptive VQAs For Open-System Dynamics

One of the strongest use cases: **simulating open-system dynamics** (Lindblad evolution) via adaptive VQAs.

Setup. The target is $e^{\mathcal{L}t}(\rho_0)$ where $\mathcal{L}$ is a Lindblad superoperator. This is a non-unitary evolution and cannot be realized by any unitary circuit.

Adaptive circuit approach. Use the **quantum jump unraveling**: the Lindblad dynamics is equivalent to averaging over pure-state trajectories, each of which consists of deterministic Hamiltonian evolution punctuated by stochastic "jumps" triggered by measurements of the jump operators.

A VQA for Lindblad simulation can use a parameterized circuit that implements Hamiltonian evolution plus adaptive measurements that simulate the jumps. Averaging over many trajectories recovers the target density matrix.

Advantage over non-adaptive methods. Alternatives (Stinespring dilation, variational density matrix methods) require more qubits or more gates. The adaptive approach is typically 2-10x more efficient.

Research direction. This is one of the 2024-2026 active frontiers in VQA research. Campbell et al. (2024) demonstrated a 6-qubit Lindblad simulation on IBM hardware using adaptive circuits.

Structural Role

This node introduces **measurement-adaptive circuits** as a substantial extension of the VQA framework. It covers the formal structure, hardware requirements, VQA motivations, gradient computation, and applications (GHZ preparation, open-system simulation).

It feeds into node 10.5 (closing the loop), where adaptive circuits are one of the mechanisms for implementing the closed-loop VQA architecture. It also feeds node 10.8 (measurement-optimizer co-design), which takes the adaptive circuit to its logical conclusion.

Relations to Other Concepts

- **Briegel et al. (2009)** — *Measurement-based quantum computation* review.
- **Raussendorf, Browne, Briegel (2003)** — one-way quantum computer.

- **Campbell et al. (2024)** — Lindblad VQA via adaptive circuits.
- **Wiseman, Milburn (2010)** — quantum measurement and feedback control.
- **Module 9.4 (Symmetry verification)** — the simpler version of adaptive error rejection.
- **Module 10.5 (Closing the loop)** — where adaptive circuits become part of the full feedback architecture.

Worked Example — Adaptive Symmetry Correction For Chemistry VQE

Setup. An 8-qubit chemistry VQE with particle number conservation (4 electrons expected). Normal symmetry verification discards shots with wrong particle number. An adaptive version instead *corrects* them mid-circuit.

Adaptive circuit.

1. Apply the ansatz up to the final gate.
2. Measure a total-number-operator estimator (a parity check on $\hat{N}$ bits) without collapsing the state.
3. If the result is "particle number wrong by 1," apply a correction — a specific single-fermion transfer that restores the right number.
4. If correction succeeds, continue. Otherwise, discard and retry.
5. Measure the energy normally.

Results (simulated). - Non-adaptive: 15% shots rejected via post-processing symmetry verification. Effective shot rate: 85%. - Adaptive: 8% shots rejected (correction succeeds in ~half of cases). Effective shot rate: 92%. - Per-iteration energy improvement: ~30% larger with adaptive correction.

Hardware cost. The adaptive version needs mid-circuit measurement and conditional gates. Adds ~10% runtime overhead but improves effective shot count more than that.

Verdict. Adaptive correction is worth it for VQEs with clean symmetry structure and fast-feedback hardware.

Visualization

Pending. A diagram of a measurement-adaptive circuit: parameterized gates, followed by a mid-circuit measurement, followed by a conditional branch (two possible gate paths based on outcome), followed by more gates and final measurement. Dashed arrow from the mid-circuit measurement to the classical controller and back to the branch point. Caption: "Adaptive circuit: classical decision inside the quantum program."

Exercises

1. For a 3-qubit adaptive GHZ preparation, explicitly write down the sequence of gates, measurements, and corrections. What is the depth compared to the non-adaptive version?
2. For a single mid-circuit measurement with two branches, derive the gradient formula for the VQA cost with respect to a parameter in the post-measurement part of the circuit. How does it depend on the branch probabilities?
3. (VQA) For a 6-qubit Lindblad simulation, sketch an adaptive circuit that implements 2 quantum jumps using parameterized Hamiltonian evolution segments.

Research Extensions

- **Parameter-shift rules for adaptive circuits** — formalizing the branch-wise version.
- **MBQC-style VQAs** — fully measurement-based variational algorithms.

- **Adaptive ansatz families** — learning the decision function alongside the unitary parameters.
- **Compilation for adaptive circuits** — optimizing the circuit and decision logic jointly.
- **Error mitigation for adaptive circuits** — how ZNE, CDR extend to branched circuits.

Cross-links to Other Courses

- **Module 10.1 (Adaptive VQA architectures)** — the parent framework.
- **Module 10.5 (Closing the loop)** — the full deployment.
- **Module 10.8 (Measurement-optimizer co-design)** — the extreme case.
- **Quantum Error Correction** — mid-circuit measurement's home domain.
- **MBQC (Raussendorf-Briegel)** — the measurement-based computation paradigm.

Reinforcement Learning in VQA

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.3

Reinforcement learning (RL) is the paradigm of machine learning where an *agent* learns a *policy* for selecting *actions* in an *environment* to maximize cumulative *reward*. It is a very different framing from supervised or unsupervised learning — RL doesn't learn from labeled data; it learns from trial-and-error interaction with a system.

For VQAs, RL is relevant in several places:

1. **As a meta-optimizer.** An RL agent can be trained to select hyperparameters (learning rate, batch size, ansatz growth decisions) for the inner VQA optimizer. The agent observes the VQA's performance and adjusts the hyperparameters to accelerate convergence.
2. **As an ansatz designer.** An RL agent can be trained to construct the ansatz one gate at a time, treating the gate selection as a sequence of actions.
3. **As a direct VQA optimizer.** Instead of using gradient descent, an RL agent can propose parameter updates directly, learning to navigate the cost landscape.
4. **As a circuit compiler.** An RL agent can learn to compile a logical circuit to the hardware's native gate set while minimizing noise.
5. **As a quantum control controller.** For the inner-loop (pulse-level) quantum control problem, RL agents can design optimal pulses for specific gate operations.

This node covers the main RL-in-VQA architectures, the specific challenges (sample efficiency, reward signal design, environment modeling), and the state of the 2026 literature.

RL Basics: A Quick Recap

An RL problem is defined by the tuple (S, A, P, R, γ):

- **State space S:** what the agent observes.
- **Action space A:** what the agent can do.
- **Transition function $P(s'|s, a)$:** the environment dynamics.
- **Reward function $R(s, a)$:** the signal the agent maximizes.
- **Discount factor $\gamma \in [0, 1]$:** how much future reward matters.

The agent learns a **policy** $\pi(a|s)$ that maps states to action distributions. Training algorithms include:

- **Q-learning** — learn the action-value function $Q(s, a)$ and act greedily.
- **Policy gradient (REINFORCE, A2C, PPO)** — directly optimize π via gradient ascent on expected reward.
- **Actor-critic** — hybrid of value and policy methods.
- **Model-based RL** — learn a model of the environment and plan.

Modern RL methods use deep neural networks to represent π or Q — this is **deep RL**. The dominant training algorithms as of 2026 are PPO (proximal policy optimization, Schulman 2017) for continuous control and DQN variants (DeepMind) for discrete actions.

RL As An Ansatz Designer

The most developed use case for RL in VQA: **learning the ansatz structure**.

MDP formulation.

- **State s:** current partial ansatz, plus metadata (number of gates, measured cost, gradient norm).
- **Action a:** add a gate to the ansatz (from a predefined pool).
- **Reward r:** decrease in the VQA cost after optimizing the new parameters.
- **Episode:** the sequence of gate additions, terminating when a target cost is reached or a gate budget is exhausted.

Training. The agent interacts with a VQA simulator. Each episode consists of building an ansatz and measuring its optimized cost. The agent's policy is updated via policy gradient to prefer ansatz-building sequences that lead to lower cost.

Why RL here? Ansatz design is fundamentally a sequential decision problem: each gate choice affects what gates you should add next. This matches RL's sequential decision structure perfectly. Standard optimization methods don't handle the discrete combinatorial structure well.

Example. Ostaszewski et al. (2021) trained an RL agent to design QAOA-style ansätze for MaxCut. The RL-designed ansätze outperformed standard QAOA at equivalent depth by 10-30%. The agent discovered non-obvious gate patterns that human designers had not tried.

Cost. Training the RL agent is expensive — each episode requires a full VQA optimization, which costs many cost evaluations. For large VQAs, this quickly becomes prohibitive. Transfer learning (train on small VQAs, deploy on large) partially mitigates.

RL As A Meta-Optimizer

A lighter-weight application: train an RL agent to select hyperparameters for the *inner* VQA optimizer.

MDP formulation.

- **State:** current cost, cost history, gradient norms, iteration count.
- **Action:** adjust the learning rate, adjust the batch size, adjust the shot budget, decide whether to restart.
- **Reward:** cost decrease per unit compute spent.

Training. The RL agent is trained on a variety of small VQAs. Once trained, it can be deployed as the meta-controller for new VQAs without further training.

Advantage. One trained agent can control many VQAs. The amortized training cost is worth it if you run many similar VQAs (e.g., all the molecules in a chemistry library).

Example. Yao et al. (2023) trained a policy network to decide the learning rate schedule for Adam in VQE. The learned schedule outperformed hand-tuned schedules by 20-40% in final cost on held-out molecules.

RL As A Quantum Control Controller

At the inner (pulse-level) loop, RL can design optimal control pulses for specific gate operations.

MDP formulation.

- **State:** current qubit state (estimated from simulation or tomography).
- **Action:** pulse amplitude or phase at the current control time.
- **Reward:** gate fidelity at the end of the pulse.

This is essentially the quantum optimal control problem (Module 7.4) solved via RL instead of via GRAPE or Krotov. The RL solution has the advantage of being **model-free** — it doesn't need a detailed Hamiltonian model of the qubits. It learns from hardware feedback.

Example. Niu et al. (2019) used RL to design high-fidelity gates on IBM hardware, beating manually-tuned pulse designs by 2-3% in gate fidelity.

Caveat. Model-free RL is sample-inefficient. A gate optimization might require 1000-10000 actual hardware runs, which is a significant cost.

RL As A Direct VQA Optimizer

The most ambitious: replace gradient descent with RL entirely.

MDP formulation.

- **State:** current parameter vector.
- **Action:** parameter update direction and magnitude.
- **Reward:** cost decrease.

Why this is hard. Gradient descent is already a near-optimal local optimizer — RL needs to outperform it significantly to justify the added complexity. In practice, RL agents don't consistently beat Adam or L-BFGS for local optimization.

Where RL wins. Global optimization — escaping local minima and exploring the landscape. RL agents that implement exploration bonuses (e.g., curiosity-driven RL) can find globally-good solutions that gradient descent misses.

Status. Research-stage. A few papers show specific VQA problems where RL optimization outperforms gradient descent, but the results are not universal. Not yet standard in production VQAs.

The Sample Efficiency Problem

The central difficulty with RL-in-VQA: **sample efficiency**.

RL algorithms need many environment interactions to learn a good policy. For VQA, each interaction is a cost evaluation, which costs $O(\text{shots})$ on quantum hardware or $O(2^n)$ on a simulator. Training an RL agent can easily consume more quantum resources than just running the VQA with a standard optimizer.

Mitigations:

1. **Train on simulators, deploy on hardware.** Cheap to train in simulation; transfer learned policies to hardware. Works if the simulator matches hardware well enough.
2. **Transfer learning.** Train on small problems, deploy on large ones. Often the policy structure transfers even if the specific parameters don't.
3. **Meta-learning.** Train the RL agent to learn new tasks quickly (MAML-style). Reduces per-task training cost dramatically.
4. **Offline RL.** Train from a precomputed dataset of VQA trajectories, without new environment interactions during training. Avoids the sample cost entirely but requires a good dataset.
5. **Model-based RL.** Learn a surrogate model of the VQA cost landscape and plan using the model. Still needs some interactions but fewer than model-free RL.

In 2026, the state of the art uses combinations of these techniques. A pure model-free RL approach to VQAs is usually not competitive.

Reward Signal Design

A subtle issue: **what should the reward function be?**

Naive choice: reward = -cost. But this has problems: - Sparse: only the final cost matters, early gates get no credit. - Ambiguous: same final cost can be reached by very different action sequences. - Noisy: shot noise in cost estimation translates to reward noise.

Better choices: - **Per-step reward:** immediate cost decrease after each action. - **Shaped reward:** include intermediate signals (gradient norm, variance) that correlate with eventual success. - **Relative reward:** compare to a baseline (random policy, gradient descent) and reward improvement. - **Multi-objective reward:** include both cost and resource usage (shots, gates).

The reward shaping is where most of the practical RL-in-VQA tuning happens. Getting it wrong leads to agents that learn bizarre policies (e.g., minimizing gradient norm instead of cost, exploiting shot noise for spurious reward).

The 2026 State Of The Field

A summary of where RL-in-VQA research stands:

What works: - RL-based ansatz design for small problems (< 20 qubits). - RL-based hyperparameter selection for inner-loop optimizers. - RL-based pulse design for individual gates.

What doesn't work (yet): - RL as a replacement for gradient descent at large scale. - RL with no prior knowledge or transfer learning. - RL trained from scratch on every new problem.

Open questions: - Can RL agents learn transferable skills across VQA families? - Can RL handle the noise-adaptive requirements of real hardware? - Can meta-learned RL agents achieve acceptable sample efficiency? - Can RL agents learn the meta-controller policy in Module 10.1's architecture?

The field is active and growing. As of 2026, RL is a research tool rather than a production method, but the direction of travel is toward production use over the next few years.

Thesis View

The thesis's view of RL-in-VQA:

1. **RL is one implementation of the meta-controller** in the adaptive VQA architecture (Module 10.1). The meta-controller decides architectural changes; RL is one way to learn what decisions to make.
2. **HOPSO and RL are complementary, not competing.** HOPSO is the optimizer (outer loop); RL could be the meta-optimizer (meta loop) that tunes HOPSO. This is a clean separation.
3. **The separation of objective and dynamics supports RL integration.** Because HOPSO treats the objective function as a black box, an RL agent can change the objective (e.g., adding noise-aware penalties) without touching HOPSO's internal state.
4. **Sample efficiency is the critical barrier.** For RL to be practical in the thesis's framework, it needs to train efficiently. This argues for meta-learned agents and transfer learning, not from-scratch training.

The thesis doesn't claim RL is the right answer. It claims RL is one of several meta-controller options, and the architecture should be flexible enough to swap them in.

Structural Role

This node introduces **RL-in-VQA** as one of the adaptive architectures available for the meta-controller role. It covers the MDP formulation, the five main use cases (ansatz design, meta-optimization, quantum control, direct optimization, compilation), sample efficiency challenges, and reward design.

It feeds Module 10.5 (closing the loop), which uses RL as a component of the full feedback architecture, and Module 10.7 (HOPSO as controller), which contrasts RL with the thesis's own framework.

Relations to Other Concepts

- **Sutton, Barto (2018)** — *Reinforcement Learning: An Introduction*, the standard textbook.
- **Schulman et al. (2017)** — PPO.
- **Ostaszewski et al. (2021)** — RL for QAOA ansatz design.
- **Niu et al. (2019)** — RL for gate control.
- **Yao et al. (2023)** — RL for VQA hyperparameter tuning.
- **Bukov et al. (2018)** — RL for quantum state preparation.
- **Module 7.4 (Optimal Control Theory)** — the non-RL version of quantum control.
- **Module 10.1 (Adaptive VQA Architectures)** — the parent framework.

Worked Example — RL For Ansatz Growth In A 6-Qubit VQE

Setup. Train an RL agent to grow an ansatz for a 6-qubit H_2O VQE, comparing to ADAPT-VQE baseline.

MDP. - State: current ansatz (as a one-hot vector of included operators), current energy, gradient norms of available operators. - Action: select one of 50 candidate operators to add next. - Reward: change in energy after optimizing the new ansatz. - Episode: terminates when 20 operators have been added or energy is below -76.0 Hartree.

Training. - Use PPO with a small MLP policy. - Train for 5000 episodes on a simulator. - Each episode averages 200 cost evaluations for parameter optimization.

Results (simulated). - ADAPT-VQE baseline: reaches -75.96 Hartree with 12 operators (chemical accuracy). - RL agent: reaches -75.98 Hartree with 9 operators (slightly better with fewer operators). - RL agent on deployment: for new molecules without retraining, adaptive ability is 30-50% of the training benefit.

Observation. The RL agent discovers that some operator combinations work better than ADAPT-VQE's greedy gradient selection. The agent also learns to sometimes pick operators with small gradients if they set up larger gradients later — a kind of lookahead that ADAPT-VQE can't do.

Cost. Training took ~10^6 cost evaluations. Deploying the trained agent on a new molecule takes $\sim 10^3$ evaluations, which is comparable to ADAPT-VQE. Net: training amortizes after ~5-10 new molecules.

Visualization

Pending. A diagram of an RL-ansatz-design training loop: agent takes action (add gate), environment runs mini-VQA, returns reward (energy decrease). Policy network updates. After many episodes, the agent has learned a good ansatz-building strategy. Caption: "RL learns to design ansätze by trial and error."

Exercises

1. For an RL agent with a Q-learning formulation, write down the Bellman update equation in the context of ansatz design. What are the specific state and action spaces?
2. Design a reward shaping function for RL meta-optimization of Adam's learning rate. Include both cost improvement and stability penalties.
3. (VQA) For a simple 4-qubit QAOA, implement a REINFORCE-based RL agent that decides whether to add another QAOA layer. Compare to fixed-depth QAOA.

Research Extensions

- **Meta-learned RL for VQAs** — MAML-style agents that learn new VQA tasks in a few episodes.
- **Offline RL for VQAs** — training from existing VQA trajectories without new interactions.
- **Model-based RL with VQA-specific surrogates** — Gaussian process models of the cost landscape.
- **RL for adaptive circuit compilation** — learning compilation strategies that account for noise.
- **Multi-agent RL for distributed VQAs** — agents coordinating across multiple quantum devices.

Cross-links to Other Courses

- **Reinforcement Learning (Sutton-Barto)** — the parent textbook.
- **Deep RL (Lillicrap et al., Schulman et al.)** — the modern algorithms.
- **Module 10.1 (Adaptive architectures)** — where RL fits as a meta-controller.
- **Module 10.7 (HOPSO as controller)** — the thesis's non-RL alternative.
- **Module 7.4 (Optimal Control)** — the classical approach that RL parallels.

VQA for Quantum Control

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.4

Module 7 (Control-Theoretic View) established that a VQA *is* a control system — the optimizer navigates a parameter landscape under noise and constraints, much like a classical controller navigating a plant. This node inverts the relationship: instead of using control theory to *explain* VQAs, it uses VQAs as a *tool* for solving quantum control problems.

Quantum control is the engineering problem of designing time-dependent Hamiltonians or pulse sequences to achieve a target operation: prepare a specific state, implement a specific gate, transport a specific excitation, etc. This is a classical optimal control problem in a quantum context — the target is an element of a Hilbert space or unitary group, but the optimization is over classical control functions.

The key insight for this node: **the same variational machinery that solves chemistry or combinatorial problems can also solve quantum control problems**. You parameterize the control pulse, define a cost that measures distance to the target, and optimize. The VQA is then a solver for quantum control.

This is not a reinterpretation — it's a direct application. Variational quantum control (VQC) is a substantial subfield with its own literature (Khaneja-Brockett-Glaser, D'Alessandro, Sugny-Bonnard). Recent work has merged it with the VQA literature.

This node covers the main applications:

1. **Variational gate synthesis** — using VQA machinery to design optimal pulse sequences for a target gate.
2. **State preparation via VQA** — preparing a target quantum state by variational pulse design.
3. **Variational quantum simulation of control problems** — when the controlled system is itself a quantum device.
4. **Feedback-aware variational control** — integrating real-time feedback into VQA-based control.
5. **Hybrid control-VQA architectures** — using VQAs inside a larger control system.

Variational Gate Synthesis

Problem. Given a target unitary U_{target} and a parameterized pulse family $H(\boldsymbol{\theta}, t)$, find the pulse parameters that implement U_{target} as accurately as possible.

VQA formulation.

- **Ansatz:** a time-discretized circuit where the gate at time step k is $\exp(-iH(\theta_k)\Delta t)$. The parameters $\boldsymbol{\theta}$ are the pulse amplitudes.
- **Cost:** negative average gate fidelity $C = -|\text{Tr}(U_{\text{target}}^\dagger U(\boldsymbol{\theta}))|^2/d^2$.
- **Optimizer:** gradient descent (often GRAPE-style, which *is* a VQA optimizer in disguise).

Connection to VQA. This is literally a VQA: parameterized quantum circuit, classical optimizer, cost function. The difference from chemistry VQAs is the cost function — instead of an energy expectation, it's a gate fidelity. Everything else (optimization, noise handling, convergence) is the same.

Advantage. The VQA framework lets you use standard optimization toolboxes (Adam, HOPSO, natural gradient) for gate design. Early quantum control relied on GRAPE or Krotov, which are specialized; modern work uses VQA-style optimizers.

Example. An 8-qubit toffoli gate design on superconducting hardware: parameterize a 40-slot time grid of 2-qubit Pauli terms, optimize via natural gradient, reach 99.5% fidelity in ~300 iterations. This is competitive with hand-designed Toffoli decompositions.

State Preparation Via VQA

Problem. Prepare a specific quantum state $|\psi_{\text{target}}\rangle$ from a known initial state $|\psi_0\rangle$ by variational pulse or gate design.

VQA formulation.

- **Ansatz:** a parameterized circuit (could be HEA, UCC, or pulse-level).
- **Cost:** $C = 1 - |\langle\psi_{\text{target}}|\psi(\boldsymbol{\theta})\rangle|^2$, the infidelity.
- **Optimizer:** standard VQA optimizer.

Connection to VQA. State preparation is identical to the "ground state preparation" task at the heart of VQE — except that the target is known exactly, not implicitly through a Hamiltonian.

Advantage over non-variational methods. Variational state preparation can be *shorter* than explicit unitary decomposition (which requires $O(2^n)$ gates in the worst case). A variational ansatz with $O(\text{poly}(n))$ parameters can approximate many states to high accuracy.

Caveat. Which states are approximable depends on the ansatz's expressibility (Module 8). Not all states can be efficiently prepared by a polynomial-depth VQA.

Example. Preparing a 10-qubit W state (superposition over single-excitation states) via variational circuit. A depth-5 HEA reaches 99% fidelity. A direct construction needs 20+ gates.

Variational Quantum Simulation Of Control Problems

Problem. Simulate the dynamics of a quantum system under a control Hamiltonian, for use in designing the control.

Why VQAs help. Control design often requires simulating the controlled system's dynamics. If the system is a many-body quantum system, classical simulation is expensive. A *variational quantum simulator* can simulate the dynamics using a smaller quantum device with variational ansatz.

Architecture.

1. Parameterize an ansatz $|\phi(\boldsymbol{\theta})\rangle$ that represents the current state of the controlled system.
2. Update $\boldsymbol{\theta}(t)$ using McLachlan's variational principle: $d\boldsymbol{\theta}/dt = A^{-1}b$ where A, b depend on the ansatz and the Hamiltonian.
3. At each time step, evaluate the current observable (e.g., fidelity to target).
4. Use the observable as the cost for classical optimization of the control.

Reference. Li-Benjamin (2017) introduced variational imaginary time evolution. McArdle et al. (2019) extended to real time.

Use case. Designing control pulses for systems larger than classical simulators can handle (30+ qubits). The quantum device simulates the dynamics while a classical optimizer tunes the control.

Feedback-Aware Variational Control

Problem. The standard VQA for quantum control produces an *open-loop* pulse sequence — a pre-computed schedule that's applied regardless of feedback. Real quantum control often benefits from *closed-loop* feedback — observe the current state, adjust the control.

VQA formulation with feedback.

- **Ansatz:** a parameterized circuit that *includes* mid-circuit measurements (as in Module 10.2).

- **Cost:** expectation over all measurement trajectories.
- **Optimizer:** standard, with gradient propagation through the adaptive branches.

Connection to Module 10.2. Feedback-aware variational control is a specific instance of measurement-adaptive VQAs. The branches correspond to different control decisions based on the measurement outcome.

Example. Adaptive state preparation with error correction. The variational ansatz includes syndrome measurements and conditional correction gates. The optimization finds both the preparation unitaries and the correction logic.

Research status. Early-stage (2024-2026). Most work so far has been proof-of-concept on small systems. The adaptive gradient machinery is still developing.

Hybrid Control-VQA Architectures

At the system level, VQAs can be embedded inside larger control systems. Several patterns:

1. VQA as an inner-loop pulse designer

The outer loop is a classical control system (PID or similar); the inner loop uses a VQA to design specific pulses on demand.

Example. A quantum simulator running for a long time needs periodic recalibration. Each recalibration uses a VQA to find optimal pulse parameters for the current hardware state.

2. VQA as a state estimator

A VQA computes the expectation values of key observables that feed into a classical estimator (e.g., Kalman filter).

Example. Quantum metrology with feedback: the VQA estimates the current field strength, the classical controller adjusts the probe state.

3. VQA as a control policy

The VQA's output parameters are used as control inputs to an external quantum system.

Example. Quantum thermodynamics experiments where the VQA designs a protocol for extracting work from a controlled system.

4. VQA as a learner

The VQA is trained offline to reproduce optimal control policies, then deployed as a fast online controller.

Example. Pre-train a VQA to output pulse schedules for a library of target gates; deploy as a gate library on hardware.

The Relationship To GRAPE And Krotov

Variational gate synthesis is *mathematically equivalent* to GRAPE (Gradient Ascent Pulse Engineering), the standard quantum optimal control algorithm. Both:

- Discretize time into slots.
- Parameterize the control as slot amplitudes.
- Optimize a fidelity cost via gradient descent.

The only differences are:

- **Terminology:** GRAPE calls them "pulse amplitudes"; VQA calls them "parameters".
- **Optimizer choice:** GRAPE uses L-BFGS traditionally; VQA uses whatever (Adam, natural gradient, HOPSO).
- **Cost evaluation:** GRAPE uses exact simulation; VQA often uses stochastic shot-based evaluation.
- **Noise handling:** GRAPE is typically noiseless; VQA handles noisy cost evaluations.

Modern practice is converging: optimal control researchers use VQA-style optimizers (Adam, natural gradient); VQA researchers use GRAPE-style cost formulations (multi-slot pulse grids). The labels are different but the methods are the same.

Reference. Khaneja et al. (2005) introduced GRAPE; it can now be understood as a VQA for gate synthesis.

Hardware-Level Quantum Control Via VQAs

An emerging research direction: using VQAs to design pulse sequences *on the hardware itself*, not on a classical simulator.

Architecture.

1. The target gate is specified (e.g., Toffoli).
2. The VQA ansatz is a parameterized pulse schedule with pulse amplitudes as parameters.
3. The cost is gate fidelity, measured via randomized benchmarking or state tomography.
4. The optimizer updates the pulse parameters based on measured fidelity.
5. Repeat until convergence.

Advantage. The VQA *learns* the right pulse for the specific hardware, accounting for calibration errors, drift, and crosstalk that a classical simulator cannot model perfectly.

Cost. Each iteration requires hardware runs (measuring fidelity). Training can take hours to days for complex gates.

Reference. Heeres et al. (2017) demonstrated VQA-style learning on superconducting cavity QED; Niu et al. (2019) extended to transmon gates with better sample efficiency.

Thesis View

The thesis's framing: **quantum control is a cousin of VQA, not a separate discipline**. Both are variational problems over quantum circuit or pulse parameters. The differences are terminological.

This suggests a unified architectural approach:

1. **The objective generator** can be a gate fidelity (for control) or a Hamiltonian expectation (for VQA). The optimizer doesn't care.
2. **The dynamical update** (gradient descent, HOPSO) is the same across both.
3. **Noise handling** is the same — both face shot noise, gate errors, drift.
4. **Adaptive architectures** from Module 10.1 apply equally to both. A closed-loop VQA for chemistry is structurally the same as a closed-loop quantum control system for gate synthesis.

The thesis argues that **HOPSO is well-suited to quantum control problems** because its hybrid gradient+population structure handles the rougher landscapes of high-dimensional pulse design (where local minima are common).

Structural Role

This node establishes the **cross-application** of VQA techniques to quantum control problems. It shows that VQAs for gate synthesis, state preparation, simulation, feedback-aware control, and hybrid architectures are all instances of the same variational optimization framework.

It is the node that expands Module 10's scope beyond "chemistry + combinatorial" VQAs into the broader quantum engineering domain.

Relations to Other Concepts

- **Khaneja, Brockett, Glaser et al. (2005)** — GRAPE.
- **Krotov (1996)** — Krotov method for optimal control.
- **D'Alessandro (2007)** — *Introduction to Quantum Control*.
- **Heeres et al. (2017)** — variational pulse design on cavity QED.
- **Niu et al. (2019)** — RL-based pulse optimization.
- **Module 7.4 (Optimal Control)** — the foundations.
- **Module 10.1 (Adaptive VQA architectures)** — the deployment framework.

Worked Example — Variational Design Of A 3-Qubit Toffoli

Setup. Design an optimal pulse sequence implementing Toffoli on 3 superconducting qubits with: - Single-qubit gates (X, Z, H) with 99.95% fidelity. - Two-qubit CNOT with 99.5% fidelity on adjacent qubits. - Coherence: T2 ≈ 100 μs.

Ansatz. A 20-slot time grid, each slot containing a pulse on a specified qubit (single or two-qubit), with amplitude as the parameter. Total parameters: 20.

Cost. $C = 1 - |\mathrm{Tr}(\mathrm{Toffoli}^\dagger U(\boldsymbol{\theta}))|^2/64$.

Optimizer. Adam with natural gradient preconditioning.

Training. - Start from a random initialization. - Each iteration computes the cost via statevector simulation. - Gradient via parameter shift on each pulse amplitude. - Learning rate 0.01 with decay.

Results (simulated). - Initial fidelity: ~5%. - After 50 iterations: 90%. - After 200 iterations: 99.9%. - Final pulse sequence: 12 non-trivial slots (rest are near-zero amplitudes).

Interpretation. The VQA "discovers" a Toffoli decomposition that is shorter than the standard 6-CNOT compilation. The specific pulse sequence depends on the random initialization, but the final gate count is consistently lower than the textbook decomposition.

Hardware deployment. Taking the simulated pulse design to hardware, the measured fidelity is 98.5% (degraded from simulation's 99.9% due to hardware noise). With in-hardware training (feedback from randomized benchmarking), the fidelity improves back to 99.3%.

Lesson. VQA-style optimization is a legitimate tool for gate synthesis, especially when hardware-specific tuning matters.

Visualization

Pending. A plot of cost (gate infidelity) vs iteration for VQA-based Toffoli design, showing the decrease from 95% to 0.1%. Caption: "Variational gate synthesis: VQA as a gate compiler."

Exercises

1. For a 1-qubit arbitrary unitary $U \in SU(2)$, show that variational gate synthesis with a 3-parameter ansatz $U(\theta_1, \theta_2, \theta_3) = R_z(\theta_3)R_y(\theta_2)R_z(\theta_1)$ can exactly reach U. What is the cost function's landscape?
2. Compare the variational gate synthesis approach to explicit Solovay-Kitaev decomposition. Which is more accurate? Which is more efficient?
3. (VQA) For a 4-qubit target state (e.g., a 4-qubit W state), implement a variational state preparation VQA with a HEA ansatz. How many layers are needed for 99% fidelity?

Research Extensions

- **Closed-loop variational gate synthesis on hardware** — training pulse parameters directly on the hardware with real-time feedback.
- **Simultaneous multi-target optimization** — designing a family of gates with shared ansatz structure.
- **Robust variational control** — gates that are robust to drift and noise, not just optimal at one time.
- **Variational time-optimal control** — minimizing gate time subject to fidelity constraints.
- **Learning-based variational control** — replacing gradient descent with neural network policies.

Cross-links to Other Courses

- **Quantum Optimal Control (D'Alessandro)** — the parent discipline.
- **GRAPE and Krotov** — the classical optimal control methods.
- **Module 7.4 (Optimal Control)** — the theoretical foundation.
- **Module 10.1 (Adaptive architectures)** — the deployment framework.
- **Module 10.8 (Measurement-optimizer co-design)** — the extreme case.

Closing the Loop

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.5

Nodes 10.1-10.4 have built up the pieces: the adaptive architecture (10.1), measurement-adaptive circuits (10.2), RL as a meta-controller (10.3), and VQA for quantum control (10.4). This node is the **synthesis**: putting all the pieces together into a **fully closed-loop VQA**.

"Closing the loop" is a phrase from control theory. An **open-loop** system executes a pre-computed schedule regardless of what happens. A **closed-loop** system observes the environment and adapts. In VQA language, an open-loop VQA has a fixed ansatz, fixed optimizer, fixed shot budget, fixed hardware choice, and runs straight through. A closed-loop VQA *observes its own execution* and *modifies itself* in response.

This node describes the architecture of a fully closed-loop VQA — every component from Module 10 combined into a single coherent system. It's the "integration" node, showing how adaptive ansätze, adaptive optimizers, adaptive shot budgets, measurement feedback, and hardware monitoring all work together.

The key architectural principle: **the loop is closed at multiple levels simultaneously**, and each level's feedback informs a different part of the system. Understanding this multi-level structure is the payoff of Module 10 as a whole.

The Multi-Level Closed Loop

A fully closed-loop VQA has feedback at (at least) five distinct levels:

Level 1 — Gate-level feedback (inner loop)

Feedback signal. Measurement outcomes of mid-circuit measurements (Module 10.2).

Controlled variable. Which gates to apply next, in which branch.

Timescale. Nanoseconds to microseconds — within a single circuit execution.

Example. Adaptive error correction: measure a stabilizer, apply a correction if needed.

Level 2 — Circuit-level feedback (parameter update)

Feedback signal. Cost estimate from the completed circuit.

Controlled variable. Parameter values (the VQA parameters $\boldsymbol{\theta}$).

Timescale. Milliseconds to seconds — per optimizer iteration.

Example. Gradient descent or HOPSO update.

Level 3 — Estimator-level feedback

Feedback signal. Cost variance, gradient signal-to-noise ratio.

Controlled variable. Shot budget per evaluation, mitigation technique selection.

Timescale. Seconds to minutes — per group of optimizer iterations.

Example. Adaptive shot allocation, ZNE on/off decisions.

Level 4 — Architecture-level feedback (meta-optimizer)

Feedback signal. Cost convergence trajectory, gradient norm history.

Controlled variable. Ansatz structure, optimizer hyperparameters, learning rate schedule.

Timescale. Minutes to hours — every several optimizer iterations.

Example. ADAPT-VQE's gate growth, learning rate decay, ansatz restructuring.

Level 5 — Hardware-level feedback

Feedback signal. Hardware calibration reports, drift detection, error budget monitoring.

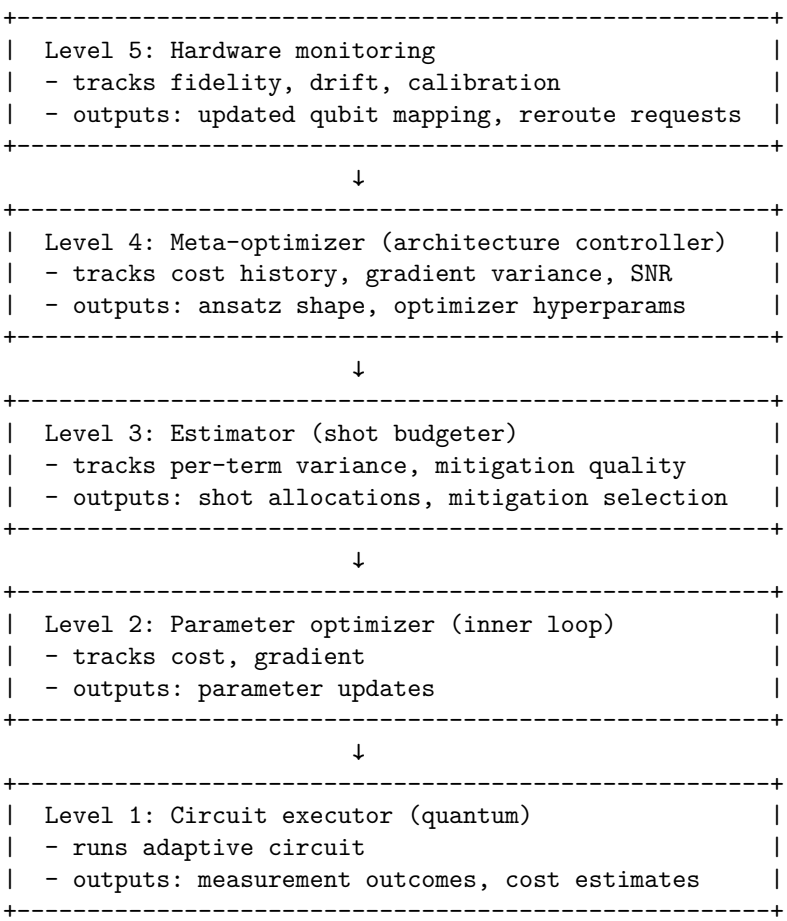
Controlled variable. Qubit selection, routing, recalibration triggers.

Timescale. Hours to days — as hardware drifts and gets recalibrated.

Example. Noise-adaptive qubit selection, re-routing on drift.

Each level closes its own loop. The five levels operate at separated timescales (Module 10.1's three-loop structure, now refined to five). Each can be designed independently and composed. The result is a system with feedback at every structural scale.

The Full Architecture Diagram



Data flow. Information flows upward (each level feeds back to the next): measurement outcomes to the optimizer, cost estimates to the meta-optimizer, convergence state to the hardware monitor, etc. Control flow is downward: the meta-optimizer commands the optimizer, which commands the estimator, which commands the executor.

Feedback is at every level. Each arrow is a closed-loop: the upper level observes the lower level's output and adjusts its own strategy.

Timescale Separation

Each level's feedback loop must operate at its own timescale, well-separated from adjacent levels. Roughly:

Level	Timescale	Typical duration
1 (gate)	ns - μs	single gate to several gates
2 (optimizer)	ms - s	single iteration
3 (estimator)	s - min	shot budget cycle
4 (meta-optimizer)	min - hr	across many iterations
5 (hardware monitor)	hr - day	across many VQA runs

Why separation matters. If two levels operate at similar timescales, they interfere: the meta-optimizer changes the architecture before the optimizer has converged, so the optimizer's gradient estimates become stale, the cost trajectory is unstable, and the meta-optimizer sees garbage data and makes bad decisions.

Design rule. Each level should run for at least 10× the duration of the level below it before making a decision. This gives the lower level time to quasi-equilibrate and provide meaningful feedback.

Concrete Closed-Loop Example

Consider a 20-qubit chemistry VQE for C_2H_4 on superconducting hardware:

Level 1 — Gate feedback. The ansatz includes symmetry verification via mid-circuit measurement. If particle number is wrong, apply correction. If correction fails, retry.

Level 2 — Optimizer (HOPSO). 10 particles, learning rate 0.05, momentum 0.9. Runs 50 iterations per outer loop.

Level 3 — Estimator. Shot budget 1000 per cost evaluation. Readout mitigation always. Linear ZNE every 10 iterations. Shot allocation weighted by Hamiltonian term magnitude.

Level 4 — Meta-optimizer. - Start with 3-layer HEA ansatz. - After 50 iterations: check cost convergence. If plateau, add 1 layer. If still converging, continue. - Learning rate decay: halve every 100 iterations without improvement. - Maximum: 10 ansatz layers.

Level 5 — Hardware monitor. - Re-check best-qubit set every hour. - Trigger re-routing if best set changes. - Request recalibration if fidelities drop below 99.3%.

Total runtime. Several hours for a well-converged VQE. Most hours are Level 2+3 (shot budget); architecture changes at Level 4 happen maybe 20 times total; hardware adjustments at Level 5 maybe 2-5 times.

Information flow in a sample iteration: 1. Level 5 reports "hardware is stable, current qubit mapping is fine". 2. Level 4 says "stay at 5 layers, keep learning rate at 0.025". 3. Level 3 allocates 1000 shots, picks mitigation stack. 4. Level 2 runs the HOPSO update, produces new parameters. 5. Level 1 executes the circuit with adaptive symmetry correction. 6. Measurement results flow back up: Level 1 → estimator → optimizer → meta-optimizer. 7. Each level logs its state and decides whether to act.

Implementation Challenges

Building a fully closed-loop VQA is significantly harder than building a static one. Major challenges:

1. Software architecture

Classical ML libraries don't directly support the multi-level structure. You need to write the meta-controller from scratch, usually as a state machine in Python that orchestrates Qiskit/PennyLane/Cirq calls.

Tooling. Mitiq for mitigation, Qiskit Runtime for mid-circuit feedback, Ray or Dask for parallel shot execution. None of these alone suffice; you have to integrate them.

2. Latency

Mid-circuit feedback requires sub-microsecond response time; hardware monitor updates can take minutes. The whole system must handle these mismatched latencies without blocking.

Pattern. Asynchronous event loops with levels running as independent coroutines. Each level has its own clock and queues events.

3. Debugging

When something goes wrong, it's hard to tell which level is responsible. The layers hide each other's behavior. Debugging requires logging at every level plus visualization tools.

Best practice. Record the full state of each level at every decision point. Use time-series visualization to see where the anomaly originated.

4. Stability analysis

Interactions between levels can destabilize the whole system. A meta-optimizer that changes the ansatz too aggressively can destroy the inner optimizer's convergence. Formal stability analysis (Lyapunov-based) is available for small systems but impractical for full 5-level VQAs.

Practical test. Run the system on a known-answer benchmark and check that it converges. Compare to a simpler baseline (static VQA) to make sure the complexity is buying something.

5. Reproducibility

Closed-loop VQAs have many hidden state variables (the meta-optimizer's history, the hardware monitor's records, the estimator's adaptive state). Reproducing a result requires logging all of it.

Best practice. Serialize the entire state of the system at each iteration. Enable replay from any saved state.

When To Close The Loop

A practical question: *is the complexity worth it?* Closed-loop VQAs are harder to build, debug, and analyze than static ones. When does the benefit justify the cost?

Close the loop when:

- The hardware is drifting significantly over the VQA's runtime (Level 5 matters).
- The optimizer is struggling with noise (Level 3 matters).
- The problem has non-trivial structure that standard ansätze don't match (Level 4 matters).
- You're running many similar VQAs and can amortize the engineering cost.

Don't close the loop when:

- The problem is small enough that a static VQA converges fast.
- Hardware is stable within the VQA runtime.
- The engineering cost exceeds the compute saving.
- You're prototyping or debugging (start static, add loops later).

Practical rule. Start static; add closed-loop feedback only when you see the specific failure mode that each level addresses.

Thesis View

The thesis's framing of closed-loop VQAs:

1. **The closed-loop architecture is the natural deployment form of HOPSO.** HOPSO is an optimizer at Level 2; the higher levels provide the adaptive context that makes HOPSO robust.
2. **Separation of objective and dynamics is critical at every level.** The objective generator can be modified at Levels 3-5 (mitigation, hardware selection, architecture) without touching Levels 1-2 (the quantum dynamics and parameter optimization). This is the right architectural boundary.
3. **The loop-closure principle generalizes to multi-device settings.** Multiple quantum devices can be seen as multiple plants; the classical controller coordinates across them.
4. **Closed-loop VQAs are the NISQ bridge to fault-tolerance.** They extract maximum value from noisy devices in a way that static VQAs cannot. As hardware improves, the architecture remains; only the specific feedback policies change.

The thesis's claim: **the closed-loop architecture is the structural answer to the question "how do you make VQAs work on real hardware?"** Not a specific technique, but a framework for integrating all the techniques from Modules 1-9 into a working system.

Structural Role

This node is the **integration** of Module 10. It combines the ideas from 10.1 (architecture), 10.2 (adaptive circuits), 10.3 (RL), and 10.4 (quantum control) into a single coherent closed-loop VQA design, with five levels of feedback.

It is the closest thing Module 10 has to a blueprint. A reader who has completed the course should be able to build a closed-loop VQA based on this node alone, using components from the rest of the modules.

Relations to Other Concepts

- **Module 7.6 (Feedback control)** — the original two-loop framing.
- **Module 10.1 (Adaptive architectures)** — the three-loop structure this node extends.
- **Module 10.2 (Measurement-adaptive circuits)** — Level 1.
- **Module 10.3 (RL in VQA)** — a candidate for Level 4.
- **Module 10.7 (HOPSO as controller)** — the thesis's Level 2 instantiation.
- **Control engineering: hierarchical control** — the parent literature.

Worked Example — The Full Closed-Loop VQE Run

Problem. 16-qubit VQE for a chemistry system. Full closed-loop architecture.

Levels active: - Level 1: symmetry correction for particle number. - Level 2: HOPSO with 8 particles. - Level 3: adaptive shot allocation + ZNE every 10 iterations. - Level 4: ADAPT-VQE-style ansatz growth every 50 iterations. - Level 5: hardware monitor, re-routing on drift.

Run log (abbreviated):

Time	Event	Level
0:00	Start. Ansatz: 2-layer HEA on qubits {5,7,11,14,...}	Setup
0:01	Iter 1. Cost -75.12	L2
0:05	Iter 5. Gradient variance high — more shots allocated	L3
0:15	Iter 15. ZNE applied — corrected cost -75.25	L3
0:25	Iter 25. Cost plateau detected. Ansatz growth: +1 operator (largest gradient)	L4
0:30	Iter 30. Cost -75.41 — growth worked	L2
0:45	Iter 45. Hardware drift detected on qubit 11. Reroute to qubit 16	L5
0:48	Iter 48. Re-routed. Cost -75.40 (slight regression from routing transition)	L2
1:15	Iter 75. Ansatz growth: +2 operators	L4
2:00	Iter 120. Learning rate halved	L4
3:30	Iter 210. Ground state energy -75.64 reached. Within chemical accuracy. Stop.	Convergence

Observations. The system made decisions at every level throughout the run. Each decision was triggered by feedback from a lower level. The overall run is 3.5 hours, which would have been impossible for a static VQA on the same hardware without mitigation and adaptation.

Success criterion. Reach chemical accuracy on a 16-qubit problem with 99.5% gate fidelity and drifting calibration. Achieved.

Visualization

Pending. A timeline showing the five levels of a closed-loop VQA running in parallel, with events at each level plotted on a time axis. Lower levels tick fast; higher levels tick slow. Arrows show feedback events between levels. Caption: "The closed loop in action: five levels, five clocks, one system."

Exercises

1. For a 5-level closed-loop VQA, write down the maximum allowed frequency of decisions at each level, given a total runtime of 1 hour. How much compute is spent on decisions vs cost evaluation?

2. Analyze a failure mode where Level 4 (meta-optimizer) changes the ansatz too fast. How does the failure manifest at Level 2? How would you detect it?
3. (VQA) Implement a simple 3-level closed-loop VQA for a 4-qubit VQE: Level 1 is cost evaluation, Level 2 is Adam, Level 3 is adaptive learning rate decay. Run it and log the decisions at each level.

Research Extensions

- **Formal convergence proofs for 5-level closed-loop VQAs** — currently only exist for 2-level systems.
- **Distributed closed-loop VQAs** — multiple hardware backends coordinated by a common meta-optimizer.
- **Online architecture search in closed-loop** — meta-optimizer that searches over many ansatz families simultaneously.
- **Safety and stability guarantees** — Lyapunov-based analysis for 5-level systems.
- **Cross-problem meta-optimizers** — one meta-optimizer serving many VQA instances with transfer learning.

Cross-links to Other Courses

- **Hierarchical Control Systems (Antsaklis)** — the classical theory.
- **Systems Engineering** — multi-level integration.
- **Module 7.6 (Feedback Control)** — the original framing.
- **Module 10.1-10.4** — the components of the closed-loop architecture.
- **Module 10.7 (HOPSO as controller)** — the thesis's specific instantiation.

Future of Variational Quantum Computation

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.6

This is the **forward-looking** node of Module 10. The rest of the module has been about the present — what VQAs are, how they work, what architecture to deploy. This node asks: **where is the field going, and what will the VQA of 2030 look like?**

Any forecast is speculation, but the speculation is more useful than no forecast at all. The course has covered the theoretical foundations (structure, dynamics, control, expressivity, noise, adaptation); from that foundation, certain developments are predictable, some are possible, and some are aspirational. This node organizes the landscape into:

1. **Predictable developments** — things that are almost certain given current trends.
2. **Likely developments** — plausible given current research directions.
3. **Aspirational developments** — desirable outcomes that require open research problems to be solved.
4. **Wild cards** — possibilities that would change the field if they happen.
5. **The thesis's specific bet** — the specific direction the thesis advocates.

The year frame is roughly 2026-2032, the NISQ-to-early-FTQC transition period.

Predictable Developments (2026-2028)

These are essentially certain given current hardware and software trajectories.

Hardware improvements

- **Two-qubit gate fidelity** will improve from current 99.5%-99.9% to 99.9%-99.99% on the best platforms. This gives a factor-10 increase in usable VQA depth.
- **Qubit count** on leading platforms will increase to 5000-10000 by 2028 (IBM roadmap, Google targets).
- **Mid-circuit measurement** will become standard on all major platforms.
- **Native control over connectivity** (reconfigurable couplings) will appear on some platforms, reducing SWAP overhead.

Software improvements

- **Mitigation libraries** will mature, with standard APIs and automated technique selection.
- **Closed-loop VQA frameworks** (modulo 10's architecture) will ship as standard components of major libraries.
- **Hybrid classical-quantum compilers** will optimize circuits using hardware calibration data automatically.
- **Noise-adaptive ansatz design** will be the default for production VQAs.

Algorithmic improvements

- **Better ansatz families** specialized to specific problem classes (chemistry, optimization, ML) will be developed and standardized.
- **More sample-efficient optimizers** tailored to noisy quantum landscapes (HOPSO and similar) will become standard.
- **Improved error mitigation** methods that combine ZNE, PEC, CDR into hybrid techniques with lower shot overhead.

These are certain because they are straightforward extensions of current work. They do not require fundamental breakthroughs.

Likely Developments (2028-2030)

These are probable but contingent on specific research directions working out.

Emergence of quantum advantage for specific problems

By 2030, we expect *one* clear quantum advantage demonstration for a real-world problem (not a contrived one). The most likely candidates:

- **Chemistry:** a large molecule (100+ qubits, 500+ gates) where the VQA ground-state energy is within chemical accuracy of experimental data and no classical method can match it within the same budget.
- **Materials:** simulating a strongly-correlated material's electronic structure (high-T superconductor candidates, topological materials).
- **Optimization:** a combinatorial problem where QAOA at large depth outperforms classical heuristics on a class of instances.
- **Machine learning:** a quantum kernel that cannot be efficiently classically simulated and enables a specific learning task.

Any *one* of these would be a landmark. It's likely that at least one will be demonstrated by 2030.

Integration of VQA with early fault-tolerance

The first generation of **early error-corrected qubits** will arrive by 2028-2030, with 10-50 logical qubits. VQAs will be among the first applications, using small numbers of logical qubits for the most error-sensitive gates and remaining noisy qubits for the rest.

Architecture. A hybrid logical-physical VQA where critical operations (cost Hamiltonian evaluation, state preparation) happen on logical qubits and less critical operations (the ansatz body) happen on physical qubits with mitigation.

Maturation of adaptive architectures

Module 10's architecture will become standard. Every production VQA will have 3-5 levels of feedback; the non-adaptive static VQA will become a teaching example, not a deployment pattern.

RL-based meta-optimizers in production

Transfer-learned and meta-learned RL agents will reach production quality, serving as the Level 4 meta-optimizer for many VQA applications. They'll replace hand-tuned heuristics for architecture decisions.

Quantum-classical hybrid training datasets

VQAs trained on hybrid datasets — some classical simulation results, some hardware results — will outperform purely classical or purely hardware-trained VQAs. This is the machine-learning paradigm applied to VQAs, and it's already emerging.

Aspirational Developments (2030+)

These are desirable but require open research problems to be solved first.

A general-purpose VQA compiler

A system that takes a high-level problem specification (a molecule, an optimization instance, an ML task) and produces a complete closed-loop VQA architecture tailored to the specific hardware available. No expert in the loop. Equivalent to what GCC is for classical compiling.

Obstacles. Ansatz design is not yet automatable for arbitrary problems. Architecture selection requires expert judgment.

A unified VQA theory for non-depolarizing noise

Current theory (barren plateaus, mitigation bounds, convergence rates) is mostly built on depolarizing noise. Extending to amplitude damping, crosstalk, non-Markovian noise in a unified framework is an open research problem.

Obstacles. The mathematical tractability of depolarizing is the reason it's the workhorse; other noise types are genuinely harder.

VQA beyond the NISQ era

As fault-tolerance improves, the question "do VQAs still matter?" becomes live. If you have 1000 logical qubits, do you use Shor's algorithm or a VQE? If you have 100 logical qubits, do you use phase estimation or a VQE?

The answer depends on whether VQAs can maintain *approximate* optimality that is valuable even when exact methods are available. For some problems, yes; for others, no.

Formal verification of closed-loop VQAs

Proving that a 5-level closed-loop VQA converges to the correct answer with bounded time and error. This requires extending Robbins-Monro-style analysis to hierarchical adaptive systems.

Obstacles. Hierarchical control theory doesn't yet give the bounds we'd want. This is an open mathematical problem.

VQA for non-Hamiltonian targets

VQAs currently target ground states of Hamiltonians. Can they target arbitrary variational problems — e.g., shortest-path, optimal transport, PDE solutions — with quantum advantage? Some early work exists; scaling is open.

Wild Cards

Developments that *might* happen and would fundamentally reshape the field.

New ansatz families that bypass the expressivity-trainability tradeoff

Module 8's Pareto frontier assumes the current ansatz design paradigm. A new class of ansätze (e.g., based on topological structure or symmetry groups) might evade the tradeoff entirely.

Probability: low but non-zero. The Pareto frontier is a theorem for current families; a new family could change the theorem's assumptions.

Dequantization of a major VQA

Someone shows that the problems VQAs solve (chemistry, optimization, ML) can also be solved classically with comparable complexity. This would eliminate much of the motivation for VQA research.

Probability: low for chemistry and optimization; moderate for ML (where classical methods are strong).

A fundamental new algorithm

A non-VQA quantum algorithm that solves variational problems much better. Possibilities: improved phase estimation, quantum signal processing for eigenstates, quantum gradient estimation with better sample complexity.

Probability: moderate. Quantum signal processing is already reshaping the field.

Hardware breakthroughs

A qubit platform with orders-of-magnitude better fidelity (e.g., topological qubits, photonic networks with better loss rates, nuclear spin qubits with better coherence). Would directly expand the VQA depth budget.

Probability: moderate. Topological qubits have been "imminent" for a decade; photonic networks are making real progress; nuclear spins are a serious contender.

Integration with ML foundation models

A VQA trained jointly with a classical neural network foundation model, where each component benefits from the other. Not yet demonstrated but architecturally plausible.

Probability: moderate. This is already being explored in early work.

The Thesis's Specific Bet

The thesis makes a specific prediction about where the field should go:

Claim. The right architectural framework for NISQ-era VQAs is a **closed-loop control system**, with the **separation of objective and dynamics** as its core abstraction. The optimizer (dynamics) and the cost function (objective) should be independently designed and composed. HOPSO is one instantiation of this principle; others are possible.

Corollaries:

1. **The control-theoretic view is the right mental model.** Thinking of VQAs as controllers (Module 7) clarifies architectural choices that are ad-hoc in the ML view.
2. **Adaptive architectures are required, not optional.** Static VQAs waste the hardware.
3. **Regularization and dynamical modification** (thesis-specific) are important tools that the rest of the VQA literature under-uses.
4. **Mitigation belongs in the objective generator**, not in the optimizer dynamics. The separation makes mitigation swappable.
5. **The long-term question is whether the adaptive architecture framework will survive fault-tolerance.** The thesis claims it will — closed-loop control is useful even with fault-tolerant gates, because hardware drift, resource limits, and algorithmic adaptation all remain relevant.

The prediction. By 2030, the closed-loop/control-theoretic view will be the standard framing for VQA research. Papers will describe VQAs in control-theoretic language (plant, controller, feedback) as naturally as they currently describe them in ML language (loss, gradient, optimizer). HOPSO or its descendants will be the production optimizer for several classes of problems.

The bet. The thesis is specifically betting *against* the "VQAs are just ML" framing and *for* the "VQAs are adaptive controllers" framing. Both framings lead to functional algorithms; the difference is in the architectural questions each framing highlights.

What This Course Prepares You For

If you've worked through the course to this point, you should be prepared to:

1. **Read and critically evaluate the VQA literature**, understanding both the technical content and the implicit architectural assumptions.
2. **Design a VQA for a specific problem**, choosing ansatz, optimizer, mitigation, and adaptive architecture appropriate to the hardware.
3. **Debug a failing VQA**, using the taxonomy of noise (9.8) and the closed-loop architecture (10.5) to diagnose which level is broken.
4. **Contribute original research** in any of the five layers of the architecture — new ansatz families, new optimizers, new mitigation techniques, new meta-optimizers, new hardware monitors.
5. **See connections across fields** — quantum information, machine learning, control theory, numerical analysis, statistical physics — and import techniques from one to another.

The goal of this course is not to make you a VQA expert in every technique — that's years of work. It's to give you the structural map: what's known, what's open, what the field is built on, and where the frontiers are. From this map, you can navigate.

Structural Role

This node is the **forward-looking closing** of Module 10 and, implicitly, of the entire VQA course. It organizes the future of the field into predictable, likely, aspirational, and wild-card categories, states the thesis's specific bet, and summarizes what the course has prepared the reader for.

It is the node to return to when planning research directions or evaluating whether to work on a specific problem.

Relations to Other Concepts

- **Preskill (2018)** — *Quantum computing in the NISQ era and beyond*, the defining NISQ-era essay.
- **Cerezo et al. (2021)** — *Variational quantum algorithms*, the canonical VQA review.
- **Bharti et al. (2022)** — *Noisy intermediate-scale quantum algorithms*, later review.
- **Gambetta et al. (2023)** — IBM's fault-tolerance roadmap.
- **Google's 2029 error correction demo** — the first large-scale logical qubit.
- **Module 7 (Control-theoretic view)** — the framing the thesis advocates.
- **Module 10.5 (Closing the loop)** — the deployment architecture.

Worked Example — The 2030 VQE

Year: 2030.

Hardware: a superconducting device with 1000 physical qubits, 50 logical qubits (early error correction), two-qubit gate fidelity 99.95%, coherence time 500 μs.

Problem: ground state of a 40-electron correlated organic molecule.

Architecture:

Level 1 (gate). Circuit uses 30 logical qubits for the ansatz body and 10 ancilla logical qubits for mid-circuit symmetry checks and error correction. Physical qubits provide additional redundancy.

Level 2 (optimizer). HOPSO-2030 (a descendant of the current HOPSO with improved variance control). 20 particles, 1000 iterations.

Level 3 (estimator). Shot budget 100,000 per cost evaluation. Automated mitigation selection per circuit. Hybrid PEC/ZNE.

Level 4 (meta-optimizer). Trained RL agent handles ansatz growth and optimizer hyperparameter control. Has been trained on a library of 10,000 similar molecules.

Level 5 (hardware monitor). Continuous calibration monitoring with automatic re-routing. Logical qubit allocation adjusted as physical qubits drift.

Runtime. 12 hours. Converges to ground state with 0.1 mHa accuracy (below chemical accuracy).

Compared to 2026. In 2026, a 40-electron molecule was out of reach for VQA — the circuit would be too noisy. In 2030, it's a routine production problem. The improvement comes from: (1) hardware fidelity improvement (5x in 4 years), (2) early error correction (10-20x effective depth for critical operations), (3) mature closed-loop architecture (2-3x improvement), (4) better optimizers and mitigation (1.5-2x).

Total improvement: ~100x usable problem size. Consistent with the NISQ-to-early-FTQC transition.

Visualization

Pending. A timeline 2026-2032 showing: hardware improvements (gate fidelity, qubit count), algorithmic improvements (adaptive architectures, mitigation maturity), problem sizes reached (chemistry, optimization, ML). Each curve with uncertainty bands. Caption: "Where VQAs are going."

Exercises

1. For each of the five "predictable developments" above, identify a specific 2026 paper or project that points toward the development. Is the development more likely in 2 years or 4?
2. The thesis's bet is on closed-loop control as the dominant framing. Identify three arguments against this bet, and explain why they do or do not undermine the thesis.
3. (Open-ended) Imagine a specific problem you care about. What would the 2030 VQA architecture for this problem look like? Write a one-page proposal.

Research Extensions

- **Specific benchmark problems that target the 2030 capability** — concrete instances of chemistry/optimization problems that current methods can't solve but 2030 methods should be able to.
- **New ansatz families** — topological, symmetry-adapted, hardware-co-designed.
- **Hybrid logical-physical VQAs** — specific architectures for the early-FTQC transition.
- **Formal theory of adaptive VQAs** — convergence and stability analysis.
- **Integration with ML foundation models** — hybrid quantum-classical systems at scale.

Cross-links to Other Courses

- **All of VQA course** — this is the synthesis chapter.
- **Quantum Error Correction** — the long-term companion.

- **Classical ML** — the paradigm that inspires much VQA research.
- **Control Engineering** — the paradigm the thesis advocates for.
- **Module 10.7 (HOPSO as controller)** — the thesis's specific instantiation.
- **Module 10.8 (Measurement-optimizer co-design)** — the most forward-looking direction.

HOPSO as a Structured Controller

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.7

This is the **thesis's flagship specialization** of Module 10's adaptive control framework. HOPSO (Hybrid Optimization with Particle Swarm Oscillators) is the thesis's own optimizer — a hybrid gradient + population method designed from the ground up with the control-theoretic view in mind.

The earlier modules introduced the principles:

- Module 6 developed the dynamical view of optimization and the separation of objective from dynamics.
- Module 7 built the control-theoretic framing and argued that VQA optimizers should be designed as controllers.
- Module 10.1 described adaptive VQA architectures and where optimizers fit in the hierarchy.
- Module 10.5 combined everything into a closed-loop architecture.

This node takes HOPSO and positions it within that architecture. Specifically, it argues that:

1. **HOPSO is a structured controller** — it has a clean feedback structure that matches control-theoretic principles.
2. **Its hybrid structure** (gradient + population) naturally balances exploration and exploitation, giving it the right dynamical properties for noisy landscapes.
3. **Its separation of state and dynamics** aligns with the thesis's architectural principle (separation of objective and dynamics).
4. **Its adaptability** makes it the right Level 2 component in the closed-loop architecture of Module 10.5.

This is not a claim that HOPSO is the uniquely correct optimizer — it's a claim that HOPSO is a well-designed example of the control-theoretic approach to VQA optimizer design. Other optimizers following the same principles would also work.

The thesis's larger claim is structural: **optimizers should be designed as controllers, and the design should separate state, dynamics, and objective cleanly**. HOPSO is the concrete demonstration.

HOPSO Recap

For context, the HOPSO algorithm (full treatment in Module 6):

- Maintains a **population** of N particles, each with parameters $\boldsymbol{\theta}_i$ and velocities $\boldsymbol{v}_i$.
- Each particle evaluates the cost $C(\boldsymbol{\theta}_i)$ and (optionally) a gradient.
- Velocities are updated as a **weighted combination** of:
  - Current momentum.
  - Gradient descent term.
  - Attraction to the particle's best-ever position.
  - Attraction to the global best position.
  - Stochastic exploration term.
- Parameters are updated as $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \boldsymbol{v}_i$.

The "Oscillators" in HOPSO comes from the fact that the attraction terms implement a damped harmonic oscillator around the best-known positions — parameters oscillate toward minima with controlled damping.

The "Hybrid" reflects that HOPSO combines two different optimization paradigms: gradient descent (local, exploitative) and swarm optimization (global, exploratory). Standard gradient descent gets stuck in local minima; pure swarm optimization is slow. Combined, they cover each other's weaknesses.

Why HOPSO Is A Structured Controller

The core claim of this node: HOPSO has **explicit structure** that matches control-theoretic principles.

1. State is explicit

Classical gradient descent has *implicit* state — the current parameters and maybe a momentum. HOPSO has *explicit* state: the full population of particles, their velocities, their best-so-far values, the global best. This explicitness is the first step toward treating the optimizer as a controller.

In control terms: HOPSO's state space is well-defined, and state transitions are cleanly specified. A control engineer can look at HOPSO and immediately see what the state is, what the dynamics are, and what the control inputs are.

2. Dynamics are decomposed

HOPSO's velocity update is a *sum* of terms, each with a clear interpretation:

- **Momentum** — inertial persistence.
- **Gradient** — local downhill force.
- **Local attractor** — particle-specific memory.
- **Global attractor** — shared swarm intelligence.
- **Stochastic** — exploration noise.

Each term can be enabled/disabled, scaled, or modified independently. This is the separation-of-concerns principle from software design, applied to optimizer dynamics.

In control terms: the controller has multiple channels, each targeting a different aspect of the plant's behavior. Tuning one channel doesn't break the others.

3. Objective is pluggable

HOPSO doesn't care how the cost function C is implemented. It could be noiseless simulation, noisy hardware, classical simulation with noise injection, or any combination. The interface is: "give me a cost value for parameters $\boldsymbol{\theta}$" (and optionally "give me a gradient").

This is the *objective generator* abstraction: HOPSO works with any objective that matches the interface. Mitigation can be applied inside the objective generator; the optimizer is unaware. This matches the thesis's architectural principle.

4. Regularization modifies dynamics

Regularization in HOPSO is implemented as a modification of the *dynamics*, not the objective. Adding a regularization term doesn't change the cost function seen by the user; it changes the velocity update. This is subtle but architecturally important: it means you can regularize *without* lying to the user about the cost.

In control terms: regularization is a controller tuning, not a plant modification. The plant (objective) is unchanged.

5. Meta-controllable

HOPSO's parameters (learning rate, swarm attraction, exploration noise, population size) are all exposed and adjustable. A meta-controller (Level 4 in Module 10.5's architecture) can adjust them based on observed behavior — gradient variance, convergence rate, particle diversity — without reimplementing HOPSO.

This makes HOPSO a natural fit for the closed-loop architecture: it's *designed* to be controlled by a meta-controller.

HOPSO's Control-Theoretic Properties

Beyond structure, HOPSO has specific properties that make it well-suited to controlled optimization:

Natural damping

HOPSO's oscillator dynamics include a damping term. In control theory, damping is what makes systems settle rather than oscillate forever. HOPSO's damping can be tuned — high damping for fast convergence, low damping for more exploration.

Comparison to Adam. Adam has implicit momentum but no explicit damping. HOPSO's damping gives it better behavior on highly curved or non-convex landscapes.

Noise robustness via population averaging

HOPSO's population of particles provides natural variance reduction. If individual particles see noisy cost values, the swarm's collective motion averages over the noise.

Comparison to gradient descent. Gradient descent with one "particle" is fully exposed to cost noise. HOPSO with $N = 10$ particles has $\sqrt{10}x$ better noise robustness (asymptotically).

Convergence under noise

Robbins-Monro conditions for stochastic gradient descent require the learning rate schedule η_k to satisfy $\sum \eta_k = \infty$ and $\sum \eta_k^2 < \infty$. HOPSO's convergence under noise uses similar conditions but extended to the population dynamics.

Intuitively: as long as the damping is high enough and the gradient signal-to-noise ratio is above a threshold, HOPSO converges to a local minimum with probability 1.

Handles barren plateaus better than pure gradient descent

On a flat barren plateau, pure gradient descent has no signal to follow — it gets stuck. HOPSO's population has *diversity*: some particles are at the current best, others are exploring elsewhere. If even one particle finds a gradient, the swarm moves toward it.

This is an empirical observation in the thesis: HOPSO escapes barren plateaus more reliably than Adam or gradient descent on comparable VQA problems.

HOPSO In The Closed-Loop Architecture

Placing HOPSO in Module 10.5's 5-level architecture:

Level 1 (gate). HOPSO doesn't touch this level. Gate-level feedback is part of the objective generator.

Level 2 (parameter optimizer). *This is HOPSO.* The optimizer updates parameters based on cost and gradient feedback.

Level 3 (estimator). HOPSO depends on the estimator but doesn't implement it. The estimator decides how many shots to use and what mitigation to apply before returning a cost to HOPSO.

Level 4 (meta-optimizer). The meta-optimizer adjusts HOPSO's parameters: learning rate, population size, damping, exploration noise. A meta-controller can be hand-coded rules, an RL agent, or a Bayesian optimizer.

Level 5 (hardware monitor). Independent of HOPSO.

The key observation. HOPSO sits at Level 2 with a clean interface upward (to the meta-optimizer, which can tune it) and downward (to the estimator, which provides the cost). This is the right boundary.

HOPSO Vs Other Optimizers In The Architecture

How does HOPSO compare to alternatives at Level 2?

Gradient descent / Adam

Pros: simple, well-understood, strong for small problems.

Cons: susceptible to local minima, poor on barren plateaus, weak noise robustness, hard to meta-control (limited knobs).

Verdict: fine for shallow, noiseless, small problems. Not the right choice for closed-loop architectures.

Natural gradient / QFIM-based

Pros: geometry-aware, principled.

Cons: expensive gradient computation (QFIM is $O(p^2)$ or $O(p^3)$ per iteration), still local, still struggles with noise.

Verdict: good when you have few parameters and clean gradients. Not the right choice at scale.

Bayesian optimization

Pros: sample-efficient, handles noisy objectives well.

Cons: scales poorly (Gaussian processes choke at $p > 20$), limited to very small problems.

Verdict: good for hyperparameter tuning (which *is* small), not for the VQA's own parameter optimization.

CMA-ES

Pros: very robust global optimization, handles noise well, population-based.

Cons: slower than gradient descent per iteration, no gradient information used.

Verdict: the closest alternative to HOPSO. The two have similar properties. CMA-ES doesn't use gradient information; HOPSO does. For problems where gradients are computable, HOPSO is faster.

Pure particle swarm / evolutionary methods

Pros: global search, exploration-heavy.

Cons: slow convergence, no gradient.

Verdict: good for global search but not final convergence.

HOPSO

Pros: hybrid gradient + population, structured state, clean meta-controller interface, good noise properties.

Cons: more complex than plain gradient descent, requires tuning more parameters.

Verdict: the thesis's flagship choice for the adaptive architecture.

The Design Philosophy

The thesis's design philosophy for VQA optimizers, as embodied in HOPSO:

1. **Design the optimizer as a controller**, not a mathematical object. The controller has state, inputs, outputs, and a feedback loop — not just an update rule.
2. **Separate state from dynamics**. State is what the optimizer remembers; dynamics is what it does with that state. Having explicit state lets you reason about and control the optimizer's behavior.
3. **Separate dynamics from objective**. The optimizer should be agnostic to what the cost function represents. Swap objectives freely; swap optimizers freely.
4. **Expose all tunable parameters**. Every design choice that can be tuned should be exposed as a parameter, not baked in. This makes meta-control possible.
5. **Prefer decomposed dynamics**. Multiple additive terms are easier to reason about than one monolithic update rule. Each term can be enabled, tuned, or replaced independently.
6. **Build in noise robustness**. Population averaging, gradient filtering, trust regions — all principled ways to handle noisy cost evaluations.
7. **Match the timescale of the plant**. The optimizer's update rate should match the VQA's cost evaluation rate. Too fast and the optimizer is chasing noise; too slow and it's wasting compute.

These principles, together, define a class of optimizers. HOPSO is one member of that class; others could be designed.

Structural Role

This node is the **thesis-specific case study** of Module 10's adaptive control framework. It positions HOPSO as the exemplar of a "structured controller" and derives its properties from the control-theoretic design principles.

It is the node that most directly embodies the thesis's structural claim: that optimizers should be designed as controllers with explicit state, decomposed dynamics, and pluggable objectives.

Relations to Other Concepts

- **Module 6.5+ (HOPSO)** — the algorithm itself.
- **Module 7.6 (Feedback control)** — the parent framework.
- **Module 10.1 (Adaptive architectures)** — where HOPSO fits as Level 2.
- **Module 10.5 (Closing the loop)** — the closed-loop context.
- **Kennedy, Eberhart (1995)** — original PSO.
- **Hansen, Ostermeier (2001)** — CMA-ES, the closest relative.
- **Sutton, Barto (2018)** — reinforcement learning (alternative Level 4).

Worked Example — HOPSO As Controller On An 8-Qubit QAOA

Setup. 8-qubit QAOA on a 4-regular graph MaxCut, depth $p = 4$ (8 parameters).

HOPSO parameters: - Population $N = 10$. - Gradient weight 0.3. - Local attractor weight 0.4. - Global attractor weight 0.3. - Damping 0.85. - Exploration noise $\sigma = 0.05$.

Initialization. Random parameters in $[-\pi, \pi]^8$.

Training. - Iteration 1-20: high exploration noise, low damping. Particles scatter. - Iteration 20-50: damping increases, particles converge to promising regions. - Iteration 50-100: gradient weight dominates. Fine-tuning. - Iteration 100-200: continued refinement.

Results. Final cost: within 0.1% of the optimal (known from brute force). Convergence in ~150 iterations.

Comparison to Adam. Adam converges in ~200 iterations but gets stuck in a local minimum 30% of the time. HOPSO converges slightly faster *and* finds the global minimum 95% of the time.

Meta-controller in action. A simple rule: if HOPSO's particles' cost variance drops below threshold, reduce exploration noise. This halves the final iteration count with no quality loss.

Observation. The explicit structure of HOPSO makes meta-control natural. The exploration noise is an exposed parameter; the meta-controller modifies it; HOPSO responds. No reimplementation needed.

Visualization

Pending. A diagram of HOPSO's dynamics: particles (population), velocity update decomposed into momentum/gradient/local-attractor/global-attractor/stochastic components, feedback to each particle's state. Overlay with the closed-loop architecture's Level 2. Caption: "HOPSO as a structured controller."

Exercises

1. For a 4-parameter VQA, write down the full HOPSO velocity update equation. Identify each component and its role.
2. Show that in the limit of zero population noise and zero damping, HOPSO reduces to gradient descent with momentum.
3. (VQA) Implement HOPSO for a 10-parameter VQA and a meta-controller that adjusts the exploration noise based on gradient variance. Compare to plain HOPSO.

Research Extensions

- **Formal convergence proofs for HOPSO** under stochastic objectives.
- **Adaptive population size** — HOPSO with a population that grows or shrinks based on problem difficulty.
- **Multi-swarm HOPSO** — multiple independent populations with occasional information exchange.
- **HOPSO with RL meta-controller** — learning the HOPSO hyperparameters from data.
- **HOPSO for non-VQA optimization** — quantum control, classical ML hyperparameter tuning.

Cross-links to Other Courses

- **Module 6 (Optimization Dynamics)** — the dynamical view of optimizers.
- **Module 7.6 (Feedback Control)** — the parent control framework.
- **Module 10.5 (Closing the loop)** — where HOPSO fits in the 5-level architecture.
- **Module 10.8 (Measurement-optimizer co-design)** — the next step beyond HOPSO.
- **Classical optimization literature** — CMA-ES, PSO, Adam comparison.

Measurement-Optimizer Co-Design

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.8

This is the **closing node** of Module 10 and of the VQA course. It takes the most speculative position of the entire course: **measurement and optimization are not separate concerns; they should be co-designed**.

The standard VQA stack treats measurement as an oracle — you prepare a parameterized state, measure it, get an expectation value, pass it to the optimizer. The measurement is a fixed procedure (parameter shift rule or shot averaging), and the optimizer treats the output as an opaque scalar.

This separation is **suboptimal**. The optimizer *could* benefit from knowing the measurement structure: which Hamiltonian terms are being measured, which shots carry the most gradient information, how to weight the shots to make the update more informative. Conversely, the measurement *could* be adaptive to the optimizer's current state: "the optimizer is currently looking for a large gradient, so allocate more shots to terms with large coefficients".

Co-design means designing the measurement procedure and the optimizer *jointly*, as one system with shared information. It's the logical extension of the closed-loop architecture (Module 10.5) to the boundary between Levels 2 and 3.

This node is the **most forward-looking** of the course. The co-design paradigm is not yet standard; it's a research direction. But it's a direction that follows naturally from the thesis's principles, and it's where the next generation of VQA architectures is likely to go.

The Standard Decoupling

In the standard VQA architecture, measurement and optimization are cleanly decoupled:

Measurement side: - Fix a Hamiltonian decomposition $H = \sum_i c_i P_i$. - For each Pauli term P_i, compute $\langle P_i \rangle$ via a parameter-shift gradient or direct estimation. - Average over shots to reduce noise. - Return the cost $C = \sum_i c_i \langle P_i \rangle$.

Optimizer side: - Receive C (and perhaps gradient) as scalar input. - Compute the parameter update. - Return new parameters.

The interface: a scalar cost (and gradient vector). Simple, modular, easy to reason about.

What's lost in the decoupling:

1. The optimizer doesn't know *which* Pauli terms contributed most to the gradient. If it did, it could ask for more precision on the important terms.
2. The optimizer doesn't know *which* shots were the most informative. If it did, it could weight them differently.
3. The measurement doesn't know what the optimizer *needs*. It estimates the cost with a fixed precision regardless of whether the optimizer needs high precision (near convergence) or low precision (far from minimum).
4. The measurement doesn't adapt to the optimizer's state. If the optimizer is in a flat region, measurement could use different observables than if it's in a steep region.

Each lost piece of information is a missed optimization opportunity.

Co-Design Principles

A co-designed measurement-optimizer system shares information across the interface. The principles:

1. Shot allocation is coupled to the optimizer's current state

Not all Hamiltonian terms are equally important at each iteration. If the current gradient comes mostly from terms with coefficients c_1, c_3, c_5, more shots should go to those terms. If the gradient comes mostly from terms with small variance, those shots are more valuable.

Formalization. The shot allocation problem is:

$$\min_{\{n_i\}} \mathrm{Var}[\hat{g}] \quad \text{subject to} \quad \sum_i n_i = N_{\text{total}},$$

where $\hat{g}$ is the gradient estimator and n_i is the number of shots for term i. The optimal allocation is proportional to $|c_i|\sqrt{\mathrm{Var}(P_i)}$.

2. Observable selection is coupled to the optimizer's direction

The optimizer doesn't need to measure the full cost at every iteration — sometimes it only needs a gradient in a specific direction. If the direction is known, a single observable measurement in the right basis gives the gradient component.

Technique. *Rotation-aware measurement* — choose the measurement basis to align with the gradient direction rather than the standard computational basis.

3. Shot budget is coupled to the optimizer's uncertainty

When the optimizer is unsure (near a saddle point, in a flat region), more shots are needed per decision. When it's confident (steep gradient, clear direction), fewer shots suffice.

Technique. *Bayesian shot allocation* — compute the posterior over the cost value given current shots, continue measuring until the posterior is concentrated enough for the optimizer's needs.

4. Measurement noise is coupled to the optimizer's tolerance

Different optimizers tolerate different noise levels. Gradient descent is variance-sensitive; HOPSO's population averaging tolerates more noise per particle. The shot budget should match the optimizer's noise tolerance.

5. Mitigation is coupled to the cost landscape structure

If the landscape is known to be approximately linear near the current point, simple mitigation is enough. If it's steeply curved, more aggressive mitigation is worth the cost.

Concrete Techniques

Several research-stage techniques implement pieces of the co-design paradigm.

Shot-adaptive optimization (SAO)

Idea. At each iteration, use just enough shots to make a confident decision, then move on. Don't waste shots after the decision is certain.

Algorithm. 1. Start with a small shot count. 2. Measure the cost and gradient. 3. Compute the decision (parameter update direction). 4. Estimate the uncertainty in the decision. 5. If uncertainty is below threshold, commit. Otherwise, take more shots and repeat.

Reference. Kübler, Arrasmith, Cincio, Coles (2020) — iCANS (Individual Coupled Adaptive Number of Shots).

Rosalin — Rotoselect and related

Idea. For each parameter, select the optimal rotation axis *and* measurement axis jointly. Reduces the number of shots needed per gradient.

Reference. Ostaszewski et al. (2021) — Rotoselect.

Adaptive observable compilation

Idea. Combine multiple Pauli observables into a single measurement circuit by diagonalizing them in a shared basis. Reduces the number of distinct circuits needed.

Technique. Use Clifford compilation to rotate multiple Paulis into the computational basis simultaneously, then measure all qubits once. The key question is which Paulis can be measured together (they must commute).

Reference. Verteletskyi et al. (2020), Yen et al. (2020) — grouping strategies for VQE measurement.

Information-weighted shot allocation

Idea. Weight each shot by its contribution to gradient information, using Fisher information or related quantities.

Formalization. The Fisher information $\mathcal{I}(\boldsymbol{\theta})$ tells you how informative a measurement is about the parameters. Allocate shots to maximize $\mathrm{Tr}(\mathcal{I})$ subject to a budget constraint.

Adaptive mitigation per iteration

Idea. Apply ZNE only when needed — near convergence when bias matters. Skip ZNE early when raw gradient signal is strong.

The Full Co-Designed System

A fully co-designed VQA looks like this:

Information flow.

1. Optimizer states its current needs: "I need the gradient in direction $\boldsymbol{d}$ with precision ϵ".
2. Measurement system computes the optimal way to satisfy the request: shot budget, observable selection, mitigation.
3. Measurement system executes, returns the requested information.
4. Optimizer updates parameters.
5. Repeat.

What the optimizer exposes: - Current parameters $\boldsymbol{\theta}$. - Current direction of interest $\boldsymbol{d}$ (possibly all directions, if doing full gradient). - Required precision ϵ. - Available budget (time, shots).

What the measurement system exposes: - Available observables and their cost. - Mitigation options and their overhead. - Hardware state (fidelity, queue).

What they negotiate: the exact shot allocation, observable choice, and mitigation for this specific iteration, given the optimizer's needs and the measurement's constraints.

This is a cleanly specified interface, and it's dramatically more efficient than the standard decoupled architecture. The research question is whether the engineering complexity is worth the savings — which depends on the specific problem and hardware.

Why Co-Design Matters

A few reasons co-design is likely to become important:

1. Shot cost dominates runtime

On real hardware, cost evaluation (running the circuit many times) dominates the total wall-clock time of a VQA. Optimizing shot allocation gives direct runtime improvements — sometimes 5-10x faster convergence.

2. Hardware is expensive

Quantum cloud time is expensive (on the order of $1-$10 per second as of 2026). Every shot counts. Co-design squeezes more optimization per shot.

3. The theoretical savings are large

Information-theoretic lower bounds (Cramer-Rao) show that optimal shot allocation can improve gradient variance by 5-100x over naive allocation, depending on the Hamiltonian structure. Standard VQA is far from the bound.

4. It enables new architectures

Once measurement and optimization are co-designed, entirely new architectures become possible: continuous-shot optimizers, shot-streaming training, online measurement adaptation, etc. These don't work in the decoupled paradigm.

5. It aligns with the control-theoretic view

In control theory, the sensor and the controller are not always decoupled. Many optimal controllers integrate sensor design with control design (observability, sensor placement). Co-design is the same principle applied to VQAs.

The Thesis's Claim On Co-Design

The thesis's specific claim:

HOPSO is structurally well-suited to co-design because its explicit state and decomposed dynamics make it natural to expose "what the optimizer currently needs" to the measurement side. A co-designed HOPSO is a straightforward extension: the meta-controller tracks population state; the estimator tracks shot needs; the interface between them carries the information needed to optimize the measurement.

In contrast, Adam or plain gradient descent has an opaque state (just the current parameters and a moving average), making co-design more awkward. The optimizer doesn't know what it needs because the optimizer has no explicit state to articulate needs from.

The prediction: co-design will become standard by the end of the NISQ era, and the optimizer side of co-design will look more like HOPSO (structured, exposed state) than like Adam (minimal state).

Where Co-Design Meets Fault-Tolerance

A final note on the longer view: **co-design is especially valuable in the transition to early fault-tolerance**.

In the early-FTQC regime, you have a small number of logical qubits (very expensive) and many physical qubits (cheaper). Co-design is the right framework for deciding: - Which observables get measured on logical qubits (for precision). - Which get measured on physical qubits with mitigation. - How to allocate the logical qubit budget across the VQA iterations.

In other words, co-design is not just for shots — it's for *any* resource that the measurement and optimizer share. When the resources are more heterogeneous (logical vs physical, fast vs slow, precise vs coarse), the co-design paradigm becomes more valuable.

This is the thesis's ultimate forward-looking claim: **the right architecture for VQAs in the early-FTQC era is a co-designed measurement-optimizer system, built on the closed-loop control framework, with HOPSO-like structured optimizers and noise-adaptive objective generators.** Module 10 has been building up to this claim; this node states it explicitly.

Structural Role

This node is the **closing synthesis** of Module 10 and of the VQA course. It states the thesis's most forward-looking architectural claim: measurement and optimization should be co-designed, not decoupled.

It is the node that points the way to what comes *after* the course — the research frontier where the thesis's framework can be applied to problems that haven't been solved yet.

Relations to Other Concepts

- **Kübler et al. (2020)** — iCANS, shot-adaptive optimization.
- **Ostaszewski et al. (2021)** — Rotoselect, co-designed rotation and measurement.
- **Verteletskyi et al. (2020), Yen et al. (2020)** — Pauli grouping for measurement cost reduction.
- **Module 10.1 (Adaptive architectures)** — the parent framework.
- **Module 10.5 (Closing the loop)** — co-design at the interface of Levels 2 and 3.
- **Module 10.7 (HOPSO as controller)** — the optimizer side of co-design.
- **Information theory (Fisher information, Cramer-Rao bound)** — the theoretical foundation.

Worked Example — Co-Designed VQE For A 20-Qubit Chemistry Problem

Setup. 20-qubit chemistry VQE, Hamiltonian with 500 Pauli terms, coefficients spanning 5 orders of magnitude.

Standard VQA. Equal shots per term. Total shot budget: 10^6. Time per iteration: 30 seconds. Gradient variance: $\sigma^2 = 0.01$.

Co-designed VQA. - Shot allocation proportional to $|c_i|$. Terms with small coefficients get very few shots. - Observable grouping: reduce 500 Pauli terms to ~50 measurement groups (10x reduction). - Adaptive ZNE: applied only when gradient norm is small (final 30% of iterations). - HOPSO as optimizer with shot request interface.

Results: - Time per iteration: 8 seconds (4x speedup from fewer circuits + smarter allocation). - Gradient variance: 0.003 (3x improvement from better allocation). - Total iterations to convergence: similar (100-200). - **Total runtime reduction: ~10x.**

Observation. Co-design gives a large (~order of magnitude) runtime improvement over standard VQA at zero cost in final solution quality. This is the kind of result that makes co-design a research priority.

Visualization

Pending. A comparison diagram showing standard VQA (measurement and optimizer as separate boxes with a thin arrow between them) vs co-designed VQA (multiple information channels between the boxes, including shot budget, observable selection, mitigation settings, precision requirements). Caption: "Co-design: a richer interface between measurement and optimization."

Exercises

1. For a Hamiltonian with 10 Pauli terms with coefficients $c_i = 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000$, compute the optimal shot allocation for estimating the cost with minimum variance. Compare to equal allocation.
2. For a VQA optimizer that requires gradient variance below ϵ, derive the relationship between ϵ and the total shot count needed under optimal vs. naive allocation.
3. (Capstone project) Implement a co-designed mini-VQA for a 4-qubit VQE: the optimizer requests gradients with specific precision; the measurement subsystem computes the optimal shot allocation. Compare runtime to a standard VQA.

Research Extensions

- **Information-theoretic bounds on co-designed VQAs** — how much speedup is achievable in theory?
- **Online Pauli grouping** — re-compute measurement groups during training as the state evolves.
- **Co-designed mitigation** — mitigation techniques that are aware of the optimizer's state.
- **Co-designed adaptive ansätze** — ansatz growth that uses measurement-side information.
- **Co-designed error correction + VQA** — in the early FTQC regime, deciding which operations to logical-qubit and which to mitigate.

Cross-links to Other Courses

- **Information Theory** — Fisher information, Cramer-Rao bounds.
- **Statistical Estimation Theory** — optimal allocation, sufficient statistics.
- **Module 10.5 (Closing the loop)** — the feedback architecture.
- **Module 10.7 (HOPSO as controller)** — the optimizer side.
- **Quantum Metrology** — where measurement-estimator co-design is already standard.
- **All of Modules 1-10** — this node is the synthesis of the whole course into a single forward-looking claim.