Module 10 — Vqa As Adaptive Control Systems

Variational Quantum Algorithms

hbar.university

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.

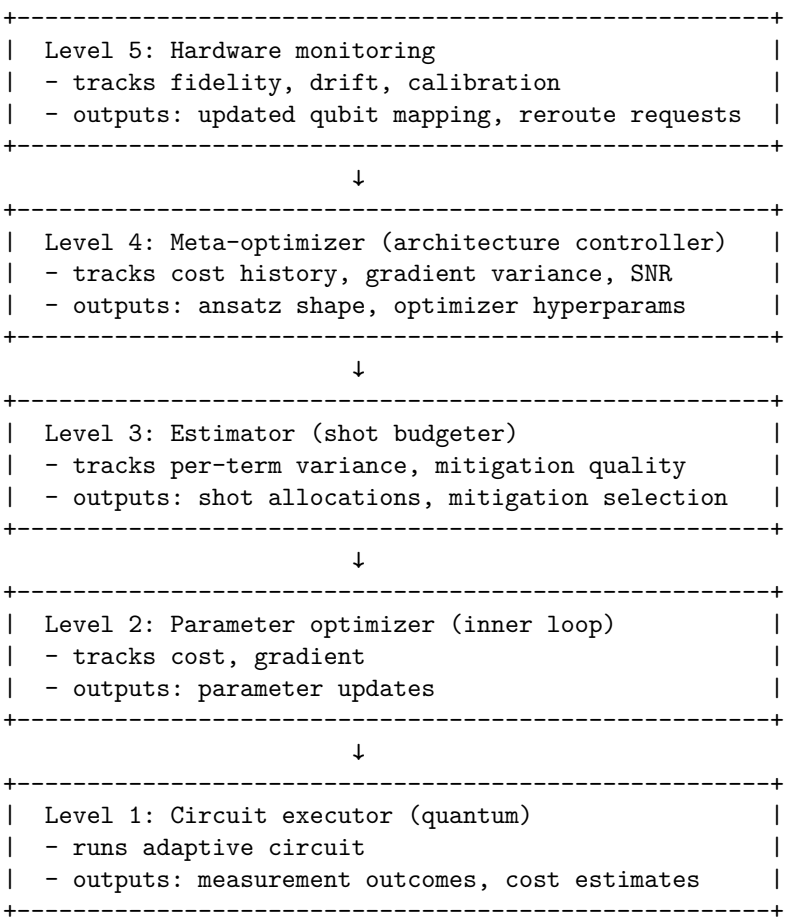
Controlled variable. Qubit selection, routing, recalibration triggers.

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.