Adaptive VQA Architectures

Variational Quantum Algorithms · Module 10 — Vqa As Adaptive Control Systems

Node 10.1

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.

Variational Quantum Algorithms · Module 10 — Vqa As Adaptive Control Systems · Node 10.1
Concepts: adaptive-systems, feedback, control-theory, parameterized-circuits
Prerequisites: 7.6, 9.6, 9.8
Enables: 10.2, 10.5, 10.7

hbar.university — own.your.cognition