Reinforcement Learning in VQA

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

Node 10.3

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.

Variational Quantum Algorithms · Module 10 — Vqa As Adaptive Control Systems · Node 10.3
Concepts: adaptive-systems, feedback, stochastic-optimization, cost-function
Prerequisites: 10.1
Enables: 10.5, 10.7

hbar.university — own.your.cognition