Sparse-Exploration Reinforcement Learning for Control of Quantum Error Correction

Jeroen Krijgsman
Responsible professor: Rihan Hai
Supervisor: Tim Littau
TU Delft

1. Drift in surface code

Useful quantum computing needs quantum error correction (QEC) to maintain achieve low error rates. One method for this, the surface code, stores one logical qubit across a $d \times d$ grid of data qubits, with ancilla qubits in between used to perform measurements to detect errors (Figure 1); a larger distance d tolerates more errors. Keeping the error rate low needs the chip's control parameters precisely calibrated, but they drift over time, forcing the program to halt for recalibration.

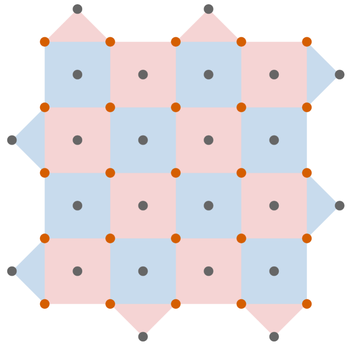

Fig 1: Distance-5 surface code. Colored dots are data qubits storing one logical qubit; gray dots are ancilla qubits that measure for errors.

2. Reinforcement Learning solution

Sivak et al.[1] avoid these halts with a reinforcement-learning (RL) agent that re-tunes the parameters online during the program (Figure 2). It reuses the error-detection signals the QEC cycle already produces from ancilla measurements as its reward, adding no extra measurements.

We independently replicate their simulation and RL agent, then build on it.

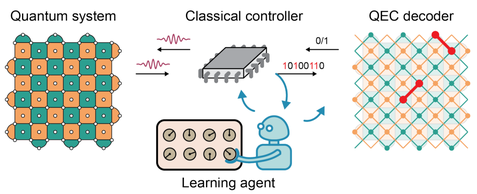

Fig 2: The RL calibration loop. Detection events from the QEC cycle feed a decoder for correction and double as the reward for the RL agent, which adjusts the chip's control parameters. Image from Sivak et al.[1]

[1] Sivak et al. Reinforcement learning control of quantum error correction, 2026 arXiv:2511.08493v3.

3. Exploration cost

To learn, the agent must explore: it slightly perturbs the parameters to find which changes help. These perturbed values are slightly miscalibrated, so they raise the error rate while training.

We refer this overhead the exploration gap (Figure 3). It is small today, but grows with code distance, longer programs, and a falling irreducible error rate, so it matters more on future hardware.

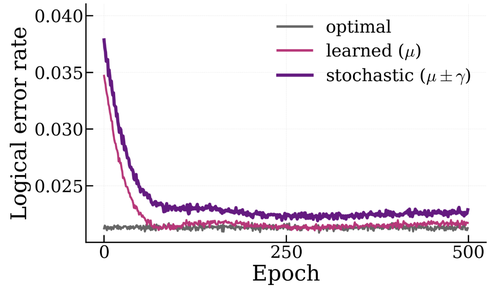

Fig 3: Logical error rate over training under mild drift. The exploring (stochastic) policy stays above the agent's own learned policy; that difference is the exploration gap.

4. Sparse exploring (our method)

How can we keep learning while exploring less? We perturb only a random subset of parameters per sample, so each is explored about once every k samples (Figure 4).

(+) Fewer perturbed parameters → lower error rate while exploring

(+) Fewer parameters share each reward → cleaner gradient

(–) Each parameter is updated less often

The second positive and negative roughly cancel out with each other under normal drift, leaving only a slight tracking loss. The first positive and slight tracking loss can be balanced against each other, giving an optimal sparsity k^*.

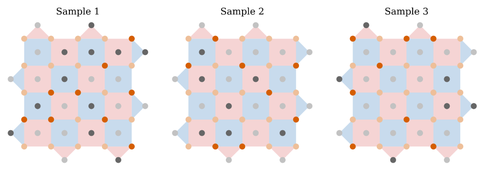

Fig 4: Sparse exploring. Each sample perturbs only a random subset of parameters, so every parameter is explored about once every k samples (CZ-gate parameters not shown).

5. Self-tuning sparsity (adaptive k)

The best k depends on the drift speed and differs per parameter. Instead of fixing it, the agent estimates k for each parameter at runtime, from quantities it already tracks (Figure 5). This adaptive $\hat{k}$ follows the per-speed optimum with no manual tuning.

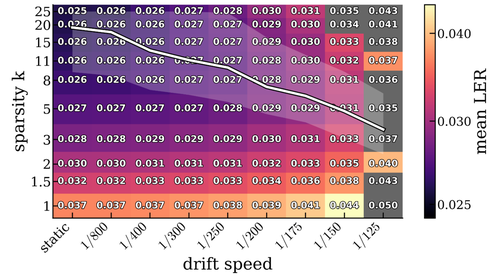

Fig 5: Steady-state LER over sparsity k and drift speed. The adaptive estimator's $\hat{k}$ (line) tracks the lowest-LER k in each column, which shrinks as drift speeds up.

6. Works under hardware-inspired drift

Does it survive realistic noise? We test against a synthetic drift whose frequencies come from real device measurements (Figure 6).

Every sparse policy sits well below dense exploration, close to the optimal-policy floor, removing most of the exploration gap.

The adaptive $\hat{k}$ solution works well for constant slow drift but can be slow to adapt to changes. Although it still performs near optimal.

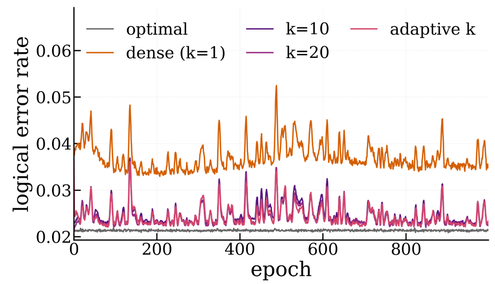

Fig 6: Logical error rate under hardware-inspired band-limited $1/f$ drift. Every sparse policy sits far below dense exploration, near the optimal-policy floor.

7. Results: robust scaling

How much of the gap closes, and does it scale? Across both sinusoidal and hardware-inspired drift, sparse exploring closes roughly 74–85% of the gap, up to about 90% for slow drift.

The reduction is essentially independent of code distance ($d = 3 - 9$) and of the irreducible error rate (Figure 7 and 8), so it carries to the large, low-error codes useful computation will need.

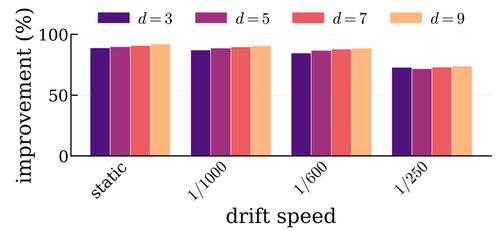

Fig 7: Gap reduction (tested under constant drift) is essentially independent of code distance ($d = 3 - 9$), staying near its ~90% ceiling except for at extreme drift.

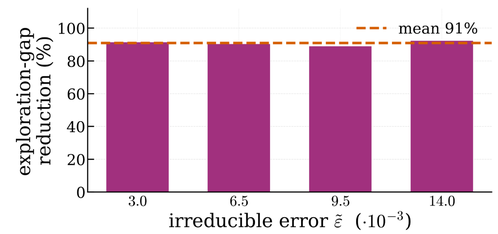

Fig 8: Gap reduction (tested under constant drift) is essentially independent of the irreducible error rate, staying near its ~90% ceiling.

7. Conclusion and future work

Perturbing only a random subset of parameters cuts the exploration gap of RL-based QEC calibration by up to ~90%, with little cost to drift tracking, and the agent tunes its own sparsity. It is a cheap add-on that works with most RL policies and pays off most on the future hardware where the gap matters most.

Future work: This work needs to be validated on a real quantum computer. With these tests on real hardware, the adaptive solution could be tuned to perform better under quick changes in drift to perform well in practice.