Sparse-Exploration Reinforcement Learning for Control of Quantum Error Correction

Shrinking the exploration gap by perturbing fewer parameters, while still tracking drift

Jeroen Krijgsman[1]
Supervisor(s): Rihan Hai[1]**, Tim Littau**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Jeroen Krijgsman[1]
Final project course: CSE3000 Research Project
Thesis committee: Rihan Hai, Tim Littau, Stephanie Wehner

Abstract

Quantum error correction (QEC) needs a quantum computer's control parameters to stay calibrated as a program runs, but these parameters drift on minute-to-hour timescales, requiring periodic recalibration. Sivak et al. [1] propose a reinforcement-learning agent that re-tunes them online, using QEC detection events as its reward and avoiding pauses for recalibration. We independently replicate their simulation and policy-gradient (PGPE) agent, and propose sparse exploring: rather than perturbing every control parameter in each sample, the agent perturbs only a random subset, lowering the mean error rate while it learns. The price of learning, referred to as the exploration gap, grows with the code distance (the size of the error-correcting code) and as the irreducible error rate (the error floor of perfectly tuned hardware) falls, and it compounds over program length. It is therefore expected to matter more on future hardware, and sparse exploring shrinks it. In simulation, under both idealized sinusoidal and hardware-inspired band-limited $1/f$ drift, sparse exploring closes roughly 74–85% of the exploration gap, reaching up to 90% for slow drift, independent of code distance (tested up to $d = 9$) and irreducible error rate. We give both a fixed-k variant and an adaptive estimator that picks the sparsity online; the adaptive variant tracks steady drift well but underperforms under sudden fast drift. The result is a cheap addition to online steering that lowers error rates over the multi-day programs useful algorithms require, pending confirmation on real hardware.

1 Introduction

Quantum computers can run algorithms that provide significant speedups over classical computers for certain useful problems [2, Ch. 4]. Shor's algorithm for breaking RSA encryption, for example, is estimated to need $\mathcal{O}(10^6)$ qubits at low error rates for about a week [3], far beyond today's $\mathcal{O}(10^2)$-qubit devices [4]. Maintaining low error rates requires precise calibration of the control parameters of the quantum computer. These control parameters drift on minute-to-hour timescales [1], dominated by slow fluctuations of two-level-system (TLS) defects [5]. The traditional solution, periodically halting the program for recalibration, does not scale well for longer programs.

A possible solution for this problem, proposed by Sivak et al. [1], is to use a reinforcement learning (RL) agent to keep control parameters calibrated during runtime of the program. This approach repurposes the measurements from quantum error correction (QEC) as a reward signal for the agent, introducing no additional qubit or measurement overhead. One disadvantage is the resulting exploration gap: since the learning process requires exploration of the parameter space, the agent must sample candidates that perform worse than its current best parameters, which raises the per-shot error rate during training. The size of this exploration gap grows with the code distance, the number of parameters per gate P, and the inverse irreducible error rate.

In this paper, we propose a method that only perturbs a sparse subset of the control parameters at each step. The intuition is that the exploration gap scales with the number of parameters being perturbed at the same time, so perturbing fewer of them at once should lower it, provided the agent can still track the drift. The research question is: *To what extent can sparse exploration reduce the exploration gap of the approach by Sivak et al. while still tracking drift, and how should the sparsity be chosen?*

Our contributions in this paper are as follows: (i) An independent replication of the simulation and RL agent used by Sivak et al., (ii) a method using sparse exploration of parameters to lower the exploration gap of the approach from Sivak et al., including a theoretical analysis, (iii) an empirical evaluation of the sparse exploration solution across different drift speeds, sparsity levels, irreducible error rates, numbers of parameters, and code distances $d \in \{3, 5, 7, 9\}$. We show this method reduces the exploration gap by roughly 75% to 85% under hardware-inspired drift, reaching the theoretical maximum of roughly 90% under slow constant drift.

Sparse exploring is related to several existing methods. The PGPE controller of Sivak et al. is a perturbation-based gradient estimator [6], in the same family as simultaneous perturbation stochastic approximation (SPSA) [7], whose convergence behavior is well studied [8]. Perturbing only part of the parameter vector also appears in gradient-free optimization, as block-coordinate descent [9], sparse perturbations [10], and structured exploration for policy gradients [11]. These methods target high-dimensional

optimization in general and do not address the drift-tracking trade-off we care about here, where perturbing fewer parameters lowers cost but slows tracking. That trade-off is the usual concern in adaptive filtering, where least-mean-squares balances steady-state error against tracking lag [12]; we make it explicit in the sparsity k. The closest work is the live RL calibration of Sivak et al. and the decoder-prior tuning of Sivak, Newman & Klimov [13], which we replicate and extend. To our knowledge, the cost of exploration itself, and its reduction by sparsity, has not been analyzed in this QEC setting.

The rest of the paper is structured as follows: First, in Section 3, the simulation and RL agent of Sivak et al. are replicated and validated against their published results. Section 4 then derives the scale of the exploration gap, and Section 5 introduces sparse exploration together with its theoretical analysis. The solution is evaluated across drift speeds, sparsity levels, and code distances in Section 6, and its transfer to real hardware is discussed in Section 7.

2 Background

Appendix A contains an introduction to quantum error correction and the surface code, the needed background knowledge this paper builds on. The remainder of this section discusses the RL solution proposed by Sivak et al. [1].

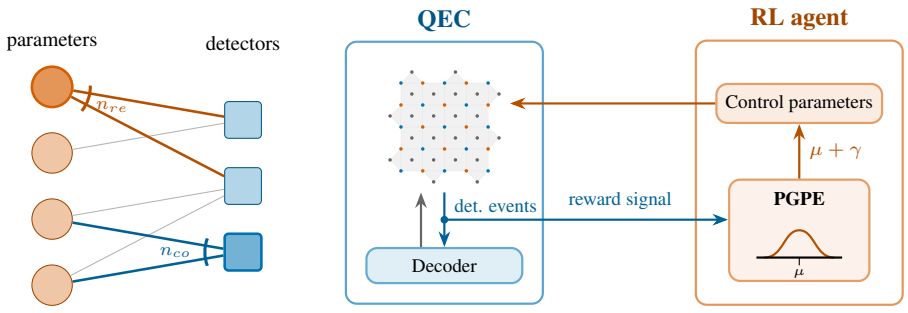
In quantum computing hardware, gates are physically implemented by applying shaped electromagnetic pulses (microwave pulses on superconducting qubits) to the qubits [14]. Because of this, the hardware has certain "physical control parameters" which have to be calibrated. These control parameters include[1] parameters for the shape and amplitude of the microwave pulses for single-qubit gates, analogous parameters for the CZ gates, and corrections for pulse distortions and phase. These parameters are calibrated in order to match the behavior of the physical qubits. However, this behavior can drift due to influence from the environment. These control parameters drift on a timescale of hours, requiring periodic recalibration [4] that is dominated by slow fluctuations of two-level-system (TLS) defects [5]. However, these periodic calibrations do not scale to the roughly week-long runtimes anticipated for cryptanalytically interesting algorithms [3]. Sivak et al. address this by using reinforcement learning (RL) to keep the control parameters calibrated online during the QEC computation; the formal model used in their simulation and replicated in this work is given in Section 3.

The number of errors of the logical qubit (the value encoded by the surface code) is measured by the logical error rate (LER) of the qubit. This is the probability of an error per QEC cycle [15]. Correspondingly, the error rate of the individual physical qubits of the surface code is the physical error rate (PER). Below the threshold, the LER decreases exponentially in the code distance d. Although the LER measures the performance of the surface code, using it as the reward for the RL has disadvantages. The LER is impractical to measure online, since it needs many QEC cycles and reading out the logical state would destroy the computation [1, Methods]. Instead, a surrogate objective is used: the average rate of error detection events in the surface code, the error detection rate (EDR). Here an error detection event refers to a parity check (a stabilizer measurement) whose outcome flips unexpectedly between rounds, a cheap signal that an error occurred nearby. Sivak et al. show that the LER and the EDR are positively correlated and share a descent direction near calibration, so steering the control parameters to reduce the EDR also reduces the LER in this regime.

Using the EDR also has another advantage: each detector [16] has its own EDR, which only depends on a small subset of the parameters [4]. To be precise, the EDR of a detector only depends on the parameters of the qubits connected to this detector region (only the four surrounding data qubits and the ancilla qubit itself). Because every reward has only a few corresponding parameters, Sivak et al. use a factor graph (Figure 1a) that links the reward signals to the parameters.[2] When computing the policy gradient, each detector only updates the control parameters it is connected to in the graph, rather than every parameter at once. Since the number of parameters per reward in this factor graph is independent of the code distance, so is the convergence speed of the RL model [1] (replicated across distances in Figure 2a).

[1] This list of control parameters is for the Google Willow chip [1].

[2] The factor graph is adopted from Appendix G of Ref. [13].



(a) Reward-routing factor graph.

(b) Continuous calibration loop.

Figure 1: System overview. **(a)** The factor graph routes each detector's reward to its local parameters; a given parameter is observed by n_{re} detectors and shares a detector with n_{co} other parameters. **(b)** The RL-based continuous calibration loop: detection events from the chip are passed to a classical decoder, which returns corrections; the same events serve as the reward signal for the RL agent, whose PGPE controller samples perturbed control parameters $\mu+\gamma$ from its Gaussian policy and applies them back to the chip, closing the loop.

A high-level overview of the RL agent can be seen in Figure 1b. The QEC block contains a standard QEC setup: the detection events are measured and transferred to a classical controller. This controller uses a decoder algorithm to attempt to reconstruct the errors that took place to give these detection events. It then uses this reconstruction to track the corrections needed to recover the original logical state at readout [15].

The control parameters for the quantum computer are perturbed slightly in order to find a gradient for the policy, as illustrated by the RL agent block on the right of Figure 1b. Sivak et al. use a Gaussian policy: every parameter has a Gaussian with a mean μ_i and standard deviation σ_i (no covariance between parameters); then random samples $\mu + \gamma$ are sampled from this Gaussian and used as control parameters. The agent uses these control parameters for a number of QEC rounds, which is used to determine the error detection rate for each of the detection zones (the reward signal). In every epoch, a batch of samples is taken from the Gaussian policy and evaluated in this way. The Gaussian parameters μ and σ are then updated using policy gradients with parameter-based exploration (PGPE) [6], which samples $\mu + \gamma$ directly in parameter space and estimates the gradients for μ_i and σ_i from the batch; the PGPE controller is described in more detail in Section 3. Intuitively, PGPE keeps a best-guess parameter vector, tries many slightly perturbed copies, keeps track of which ones lowered the error, and shifts the best guess toward them.

The policy's μ corresponds to the best estimate of the optimal parameters, while the standard deviation σ determines how much the agent explores. Because of the perturbation of the parameters, the parameters are set to slightly miscalibrated values. Because of this, the samples have a higher EDR than the agent would have using μ, and hence a higher LER (since EDR and LER move together, a higher EDR means a higher LER). This effect scales with the number of perturbed parameters per gate and with σ, which itself must grow as the drift speed grows. This inefficiency, which we refer to as the *exploration gap*, is mentioned by Sivak et al.; reducing it is the focus of this paper, with the formal treatment in Section 4.

3 Replication of Sivak et al.

In order to verify the findings in this paper, we implemented an independent replication of the simulation and RL model used by Sivak et al. [1]. This also provides a public reference for future replicators and ensures our sparse-exploring additions fit the existing code.

3.1 Simulation and RL implementation

The simulation is implemented using Python, Stim [17], and uses PyMatching [18, 19] as the decoder; the exact software versions, license, and random seed are listed in Section 9. The code for our replication is

made open source and can be found online.[3] The simulation uses an XZZX surface code [20] as the error correction method, which can be instantiated with variable distance and with measurement basis X or Z. The exact design of the XZZX surface code is based on the Stim model published in the supplementary materials of Sivak et al. This Stim model uses an SI1000 noise model [21, 1] for the reference circuit and its validation against Sivak et al. (Section 3.2); the steering and exploration-gap experiments instead use the simplified per-gate depolarizing parameter model described next. In the simulation, the physical control parameters of Section 2 are simplified: every single-qubit gate and CZ gate has a fixed number of parameters P. Each of the P parameters of a gate plays the same role, contributing equally to its depolarization rate. This rate then determines the probability of a random error[4] for this gate.

Sivak et al. use two different error models, depending on the experiment: when evaluating the ability to track drifting error, they use $\varepsilon_i = \tilde{\varepsilon}_i + \Omega_i(p_i - p_i^{opt})^2$. Here, $\tilde{\varepsilon}_i$ is the irreducible error rate (the error under optimal calibration), Ω_i is the "sensitivity" of the gate and p_i^{opt} is the optimal value of the parameter which drifts according to $p_i^{opt}(t) = \sin(2\pi f t)$. The frequency of the drift, f, is chosen equally for all parameters and can be tweaked to simulate different drift speeds.

When evaluating the convergence speed of the simulation for different numbers of parameters per gate P, it is defined as $\varepsilon_i = \tilde{\varepsilon}_i + \sum_j \Omega_{i,j} p_{i,j}^2$ with $j = 1, \ldots, P$. In calculations in Section 5.1, a combination of these two models, $\varepsilon_i = \tilde{\varepsilon}_i + \sum_j \Omega_{i,j}(p_{i,j} - p_{i,j}^{opt})^2$, is used in order to analyze both error models.

In both of these models, ε and Ω_i are chosen to match the resulting errors in Sivak et al.; however, the scale of these values differs per experimental setup. The scaling experiment, for which the results are shown in Figures 5b and 5c of Sivak et al., assumes a lowered irreducible error rate compared to the regular drifting experiment. This choice was most likely made to demonstrate the reduction of LER performance since the PER is below the threshold. Since Sivak et al. did not specify the exact choice of ε and Ω_i, we chose to sample them from a uniform distribution, with a mean chosen empirically based on their published figures.

The RL agent is the PGPE controller of Sivak et al., replicated with the stabilization techniques they use (PPO, entropy regularization, and a replay buffer) and the optimizer and symmetric-sampling choices of Sivak, Newman & Klimov [13]. Appendix D.1 describes the full setup and the one component not needed in our simplified model (detector-sensitivity rescaling). Appendix H gives the hyperparameters used.

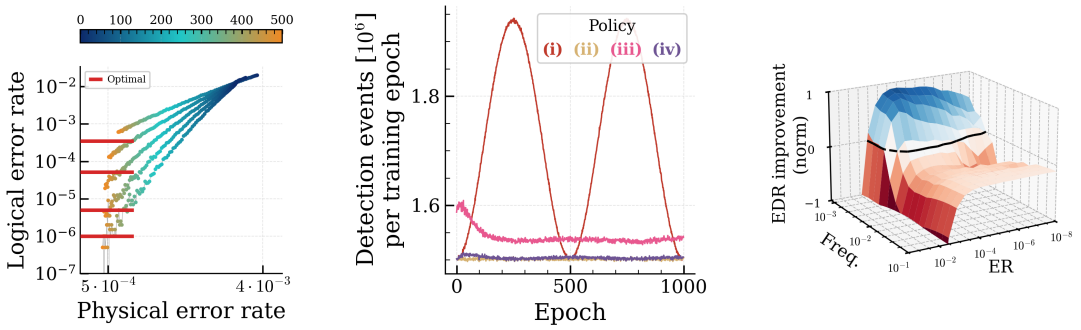
The metrics used throughout this paper are all calculated in the same manner as Sivak et al. The error detection rate (EDR) is the mean rate of error detection events in an epoch, which is the mean of all rewards. The physical error rate (PER) is the per gate mean error rate of individual qubits, which is calculated in the simulation by taking the mean error probability over the DEM's error mechanisms. Finally, the logical error rate (LER) is the probability, per QEC cycle, that the decoder fails to recover the encoded logical qubit; it is measured over a shot of N cycles and converted to a per-cycle rate via $p_L = \frac{1}{2}\left(1 - (1 - 2p_{\text{shot}})^{1/N}\right)$ [1, 4].

3.2 Validation against published results

The correctness of the implementation is verified in a few ways: first, every individual part is tested. One particular example is that the Stim XZZX surface code with the SI1000 error model is verified against the reference Stim model it is based on. All tests are included in the source code. Second, the model is verified against some simulation plots provided by Sivak et al. These replications are shown in Figure 2, with an additional plot being shown in Appendix G. From Figure 2, we can see the general behavior of the model is the same as the original plots [1]. Figure 2a and Figure 2b show near identical behavior, with the replicated LER following the same power-law scaling in the PER as Sivak et al. Figure 2a does show a slightly lower LER for a given PER compared to Sivak et al., because a fresh detector error model (DEM) is calibrated for matching each epoch rather than reusing a fixed, precalibrated DEM; since decoder calibration is not the focus of this paper, this does not affect the conclusions drawn from the

[3] https://github.com/InfiniData-Lab/qec-sparse-exploration

[4] All error types are assumed equally likely.



(a) Figure 5b (b) Figure S8a (c) Figure S8c

Figure 2: Replication of key results from Sivak et al.: panels **(a)–(c)** reproduce Figs. 5b, S8a and S8c respectively, using the independent implementation of this paper. Panel **(a)** shows the scalability of RL control of large surface codes. The simulation uses $P = 30$ parameters per gate and is evaluated for distances 3, 5, 7 and 9 (top to bottom). The red bars show the LER under optimal policy, and points are coloured by training epoch (top colorbar). Panel **(b)** shows the performance of the steering at a drift frequency of $f = 1/(1000 \text{ epochs})$ evaluated with multiple policies: (i) fixed, (ii) optimal, (iii) stochastic, (iv) learned. Here the learned policy is μ_i and the stochastic policy is $\mu \pm \gamma$. Panel **(c)** shows when the stochastic-policy steering improves EDR over the fixed policy for different simulation parameters; values greater than 0 correspond to improvement. Steering becomes worthwhile only once the drift is slow enough: the crossover (the EDR improvement at the best entropy-regularization passing 0) sits at a drift period of roughly 250 epochs (slightly higher than Sivak et al., but comparable order of magnitude), and the gain grows steadily as the drift slows further (reaching about $+0.9$ at $1/1000$). The learned-policy counterpart (Fig. S8d) is reproduced in Appendix G.

replication. Figure 2c and Figure 11 are also mostly similar, with one exception: the EDR improvement does not worsen as fast for lower values of the ER (additionally, the optimal value for the ER coefficient is lower). The reason for this is that our implementation of the σ gradient already contains a balancing term, preventing σ from collapsing to near 0 values. This difference does not worsen performance, so the model remains usable for further research.

4 The exploration gap

Figure 2b shows the performance in steering of multiple policies. Policy **(iv)** shows the performance of the learned policy μ, what the controller currently believes is the optimal calibration of the parameters. However, the controller is not able to use this policy, as it would not be able to account for drift. Instead, it must perturb the parameters slightly, giving the stochastic policy **(iii)** $(\mu+\gamma)$. This slight noise added to the policy allows the controller to estimate the gradient of μ. Because the agent is required to keep perturbing the control parameters, it uses slightly miscalibrated values. This adds a slight performance overhead to the solution, which we will refer to as the exploration gap. Sivak et al. [1] mention this exploration gap as a problem with this approach. They conclude the trade-off is worth it, since the steering with the exploration gap still outperforms the fixed policy. However, as will be shown in this section, the relative value of this exploration gap increases with larger code distances and smaller irreducible error rates. The relative value increases roughly linearly with $d + 1$, and useful algorithms demand larger code distances [3]. Additionally, as hardware improves, the irreducible error rate $\tilde{\varepsilon}$ has steadily fallen with each hardware generation [4]. This causes the exploration gap to be a relatively bigger part of the LER as hardware advances. This is further amplified by the accumulation of logical errors over QEC cycles. The whole-run success probability is roughly $(1 - p_L)^N$ over N cycles, so over the billions of cycles in a days-to-weeks run [3], a slightly worse per-cycle LER compounds into a large increase in the probability of a logical error. Sivak et al.'s exploration/accuracy trade-off is favorable on today's hardware, but the exploration gap will become a growing share of the total errors as devices approach the scale at which they become useful. This argument is further explored in Section 7.

4.1 Scale of the exploration gap

In order to derive the magnitude of this exploration gap, a few assumptions will be made. First, the following theoretical error model will be used: $\varepsilon_i = \tilde{\varepsilon}_i + \sum_j \Omega_{i,j}(p_{i,j} - p_{i,j}^{opt})^2$, where $\tilde{\varepsilon}_i$ is the irreducible error (the floor error of a perfectly tuned gate), $\Omega_{i,j}$ is the sensitivity of parameter j (how sharply error rises when it is mistuned), and $p_{i,j}^{opt}$ is its optimal value. This error model incorporates both error models used by Sivak et al. and is assumed to be a good approximation of the actual error.[5] For brevity, we write $\delta_{i,j} = p_{i,j} - p_{i,j}^{opt}$ for the miscalibration of a parameter, giving $\varepsilon_i = \tilde{\varepsilon}_i + \sum_j \Omega_{i,j}\,\delta_{i,j}^2$. This also means it is assumed the only type of error on the qubits is depolarizing error, where all errors are equally likely. The second assumption is that the LER only depends on the mean error, rather than the exact distribution of errors across the surface code. In actuality, the LER does depend slightly on how the errors are distributed across the gates, but this dependence is small relative to its dependence on the mean error. This will be empirically verified in Appendix E. With this assumption, we replace every parameter by its mean, written simply $\tilde{\varepsilon}$, Ω and σ, so the sum over the P control parameters per gate becomes a factor P.

The final assumption is that the relation between ε and p_L (the LER), scales according to the power law [15]. To be precise, we assume $p_L \sim (\varepsilon/\varepsilon_{th})^{(d+1)/2}$ for odd d, where ε_{th} is the error threshold for this implementation of the surface code. This relation has been shown by Fowler et al. [15] to be roughly accurate for normal surface codes. This assumption will be verified empirically in Section 4.2.

Under these assumptions, we can find the constants c and ε_{th} such that $p_L \approx c \cdot (\varepsilon/\varepsilon_{th})^{(d+1)/2}$. Using this relation, we find the increase in LER caused by the exploring policy, the exploration gap, to be equal to $\Delta p_L \approx c \cdot \Delta((\tilde{\varepsilon} + \sum_j \Omega\delta^2)/\varepsilon_{th})^{(d+1)/2}$. Here, p is sampled from $\mathcal{N}(\mu, \sigma^2)$, so δ is distributed as $\mathcal{N}(\mu - p^{opt}, \sigma^2)$, where we will take all δ_i and all σ_i to be equivalent. This is backed by the assumption of independence of exact distribution of errors. When near the optimal policy, we take $\mu \equiv p^{opt}$, giving a distribution for δ roughly equal to $\mathcal{N}(0, \sigma^2)$. Given that $\mathbb{E}[\delta^2] = \mathrm{Var}[\delta] + (\mathbb{E}[\delta])^2$ and $\mathbb{E}[\delta] = 0$, we find $\mathbb{E}[\delta^2] = \sigma^2$.

$$\Delta p_L \;=\; \mathbb{E}_{p\sim\mathcal{N}(p^{opt},\,\sigma^2)}[\,p_L\,] \;-\; p_L\big|_{p\equiv p^{opt}} \;\approx\; \frac{c}{\varepsilon_{th}^{(d+1)/2}}\left((\tilde{\varepsilon} + P\,\Omega\,\sigma^2)^{(d+1)/2} - \tilde{\varepsilon}^{(d+1)/2}\right) \tag{1}$$

We can now define the exploration gap Δp_L to be the difference between the expected LER near the optimal policy and the actual LER at the optimal policy. By the second assumption, the LER depends only on the mean error rate. Since this mean averages over hundreds of perturbed parameters, we can substitute $\mathbb{E}[\delta^2] = \sigma^2$ directly to get Eq. 1. Here, $\mathbb{E}[p_L|_\varepsilon] \approx p_L|_{\mathbb{E}[\varepsilon]}$ is used because ε is averaged over many parameters, so $\varepsilon \approx \mathbb{E}[\varepsilon]$. Like the power-law fit and the small-σ expansion below, this is a leading-order approximation rather than an exact identity. Near convergence, where $P\,\Omega\sigma^2 \ll \tilde{\varepsilon}$, we can expand into Eq. 2, where we find the relative cost of exploration to be roughly linear in σ^2.

$$\frac{\Delta p_L}{p_L(0)} \;\approx\; \frac{d+1}{2} \cdot \frac{P\,\Omega\,\sigma^2}{\tilde{\varepsilon}}. \tag{2}$$

With this we can conclude that, under the assumption that which parameters have higher error rates does not significantly affect the LER (verified in Appendix E), the relation between the exploration gap when near the optimal policy ($P\,\Omega\,\sigma^2 \ll \tilde{\varepsilon}$) and the average σ^2 is roughly linear. This will be verified empirically in Section 4.2.

4.2 Empirical confirmation

First, Figure 12a (Appendix G) confirms the power law [15] for this implementation of the XZZX surface code. The LER closely follows the power law, with the found relation $(c, \varepsilon_{th}) = (0.026,\ 6.15 \cdot 10^{-3})$ being close to the $(0.03,\ 5.7 \cdot 10^{-3})$ reported by Fowler et al. [15].[6]

[5] This is trivially the case for the simulation, but is also assumed for real errors.

[6] The small offset is expected, as their values are for a different code and noise model.

Next, Figure 12b and Figure 12c show Eq. 1 and Eq. 2 compared to the empirically measured relative exploration gap. It can be seen that Eq. 1 gives a close approximation for the actual value across all values of σ and all parameters. For small values of σ, the linear approximation (Eq. 2) is accurate, and lower-bounds the gap for higher values of σ. The gap's predicted inverse dependence on the irreducible error rate is also confirmed: the relative gap grows roughly linearly with $1/\tilde{\varepsilon}$.

5 Sparse exploring via random masking

In order to reduce the exploration gap derived in Section 4, we propose sparse exploring of the control parameters: rather than perturbing every parameter in each sample, each parameter is perturbed on average only once every k samples. The main benefit is a lower LER: perturbing fewer parameters at a time lowers the mean error rate, which directly lowers the LER during exploration. The cost is a potential loss of tracking ability, since each parameter is now explored less often. This cost is small, however: because fewer parameters are perturbed per reward, the variance in the gradient also drops, and these two effects roughly cancel out (shown in Section 5.1). Since $k = 1$ is equal to traditional dense exploration where all parameters are perturbed, k can be chosen such that the reduction in LER outweighs the small tracking cost. With this solution, we find the exploration gap can be reduced up to roughly 90% for steady-state performance (Section 6).

When sampling the γ for a given epoch with batch size M, we draw $M_{1/2} = M/2$ symmetric perturbation pairs; per parameter we determine a value of $\hat{k}_i$, the sparsity for this parameter.[7] Then rather than sampling $M_{1/2}$ distinct pairs of $\mu_i \pm \gamma_i$, we will sample only $M_{1/2}/\hat{k}_i$ of these pairs, leaving the rest as μ_i. The pairs to perturb are chosen randomly per parameter. This leans on the assumption that the spatial choice of which parameters to perturb barely affects the LER (so a random choice is fine); but it does ensure every parameter is perturbed about equally often with as few neighboring parameters as possible being perturbed, which minimizes tracking loss (argued in Appendix E). The full per-epoch procedure and a dense-versus-sparse schematic of the masking are given in Appendix D (Figure 8).

5.1 Theoretical analysis

The concern is that perturbing each parameter only $1/k$ as often could make the gradient up to k times noisier. We show the increase is in practice far smaller than k, because the largest part of the gradient noise comes from neighboring parameters and is left unchanged by the masking. In order to estimate by how much the convergence of the PGPE controller is affected by the sparse exploring strategy, the variance of the PGPE gradient under the error model of Section 4.1 will be analyzed. An approximation of this gradient, which is derived in Appendix B, can be found in Eq. 3. This equation gives an approximation of the variance in the gradient for $M_{1/2}$ different pairs of symmetric samples. In this approximation, the self term indicates the variance introduced by sampling the parameter itself. The coupling term indicates the variance introduced by other parameters affecting this parameter. Here, n_{co} is the number of other parameters that share a detector with parameter i.[8] Its value depends only on the position of the gate within the code, not on the code distance: including the extra weight of parameters sharing multiple rewards, $5P - 1 \leq n_{co} \leq 56P - 16$,[9] with $9P - 1$ the most common value. Finally, the measure term indicates the noise from the measurement of the rewards with $n_{re,i}$ the number of detectors observing parameter i and s^2 the per-detector measurement-noise variance (Appendix B). Here $\bar{\delta}_i = \mu_i - p_i^{opt}$ denotes the miscalibration of the policy mean; unlike δ of Section 4.1, the miscalibration of an applied parameter, it excludes the applied perturbation. The coefficient $c_{r,i}$ links the error of gate i to its detection rate, with the bare c_r in the coupling term its common value across parameters.

$$\mathrm{Var}(\nabla_{\mu_i} J) \approx \frac{1}{M_{1/2}} \Big(\underbrace{8\, n_{re,i}^2\, c_{r,i}^2 \Omega_i^2 \bar{\delta}_i^2}_{\text{self}} + \underbrace{4\, n_{co}\, c_r^2 \Omega^2 \bar{\delta}^2}_{\text{coupling}} + \underbrace{n_{re,i}\, s^2/(2\sigma_i^2)}_{\text{measure}} \Big) \tag{3}$$

[7] It is optimal to choose different values of k for different parameters, as will be derived in Section 5.1.

[8] This is for dense exploration; under masking with sparsity k, roughly n_{co}/k of these are perturbed in any given pair.

[9] Parameters can count multiple times based on the number of shared rewards.

We now introduce the sparse exploration, where every parameter is perturbed only about $M_{1/2}/k$ times.[10] The sparse exploration changes the factor $1/M_{1/2}$ to a factor of $k/M_{1/2}$. On the other hand, only about n_{co}/k parameters are perturbed at any given time. We can split the gradient (Eq. 4) into a part which is not affected by other parameters (A) and a part which is affected by the other parameters (B).

$$A = \underbrace{8\, n_{re,i}^2\, c_{r,i}^2 \Omega_i^2 \bar{\delta}_i^2}_{\text{self}} + \underbrace{n_{re,i}\, s^2/(2\sigma_i^2)}_{\text{measure}} \qquad B = \underbrace{4\, n_{co}\, c_r^2 \Omega^2 \bar{\delta}^2}_{\text{coupling}} \tag{4}$$

These parts can be used to simplify the unmasked gradient variance into $\mathrm{Var}_{\text{dense}} \approx (A+B)/M_{1/2}$ and the masked gradient variance into $\mathrm{Var}_{\text{sparse}} \approx (kA+B)/M_{1/2}$. The ratio of these variances, Eq. 5, shows that the increase in gradient variance depends on the ratio between A and B. If $kA \ll B$, the ratio is closer to 1, while $A \gg B$ gives a ratio closer to k. Since higher P gives higher B/A, we find higher P in general gives a relatively smaller increase in variance per parameter. Eq. 5 is verified in Section 5.3.

$$\frac{\mathrm{Var}_{\text{sparse}}}{\mathrm{Var}_{\text{dense}}} \approx \frac{kA+B}{A+B} \tag{5}$$

To determine when sparse exploring is worth its cost, all three effects (exploration, gradient noise and lag) can be combined into one estimate of the expected error. The miscalibration of an applied parameter p_i splits into three parts:

$$\delta_i \;=\; p_i - p_i^{opt} \;=\; \underbrace{(p_i - \mu_i)}_{\text{exploration}} \;+\; \underbrace{(\mu_i - \mathbb{E}[\mu_i])}_{\text{gradient noise}} \;+\; \underbrace{(\mathbb{E}[\mu_i] - p_i^{opt})}_{\text{lag}} \tag{6}$$

The exploration part is the applied perturbation ($\pm\gamma_i$, or 0 when masked out). The gradient-noise part is caused by the noise in the gradient estimates from sampling. The lag is the error caused by drift of the optimum [12]. Since the gradient estimate is unbiased, the update is linear in the gradient ($p_{next} = p_{prev} - \eta \cdot \text{grad} + \text{drift}$), and the true gradient is itself linear in the miscalibration (quadratic error model), the lag does not depend on the gradient variance.

The exploration and gradient-noise parts both have mean 0, and all three parts are mutually independent. Squaring Eq. 6 and taking the expectation therefore leaves only the three squared terms:

$$\mathbb{E}[\delta_i^2] \;=\; \mathbb{E}[\gamma_i^2] \;+\; \mathrm{Var}(\mu_i) \;+\; \text{lag}^2 \tag{7}$$

For a controller that processes the gradient estimates linearly, the deviation of the policy mean is proportional to the gradient variance [12]. We write this as $\mathrm{Var}(\mu_i) = \kappa\, \mathrm{Var}(\nabla_{\mu_i} J)$, where the noise gain κ absorbs both the controller details (learning rate, replay buffer) and the $1/M_{1/2}$ of the batch average in Eq. 3, and is identical for dense and sparse exploration. Due to the usage of the Adam optimizer, this κ is actually not constant, which is discussed in Section 5.2.

Filling in both strategies (dense: $\mathbb{E}[\gamma^2] = \sigma^2$ and gradient variance $A+B$; sparse: each parameter is perturbed only once every k samples, so on average $\mathbb{E}[\gamma^2] = \sigma^2/k$, with gradient variance $kA+B$):

$$\begin{aligned} \mathbb{E}[\delta_i^2]_{\text{dense}} &\approx \sigma^2 + \kappa\,(A+B) + \text{lag}^2 \\ \mathbb{E}[\delta_i^2]_{\text{sparse}} &\approx \sigma^2/k + \kappa\,(kA+B) + \text{lag}^2 \end{aligned} \tag{8}$$

To compare the benefit in the same situation, A and B are assumed equal. The lag is equal for a constant controller gain, but not under the Adam optimizer. Adam shrinks the effective step when the gradient is noisier, making the lag k-dependent. For now the lag is taken as equal, but this is calculated properly in Section 5.2. Under these assumptions, both the lag and the κB term drop out, and multiplying by Ω gives the net benefit of sparse exploring per parameter:

$$\Delta\varepsilon_{\text{benefit}} \;=\; \Omega\left(\sigma^2\,(1-1/k) \;-\; \kappa\,(k-1)\,A\right) \tag{9}$$

[10] We do not assume k to be an integer; this will be used in Section 5.2.

The first term is the exploration saving of Section 4.1; the second is the gradient-noise cost. Sparse exploring is therefore beneficial exactly when $\sigma^2/k > \kappa A$, and since Eq. 9 is concave in k, the benefit is maximized at:

$$k^* = \sqrt{\frac{\sigma^2}{\kappa\, A}} \tag{10}$$

Substituting the optimum $k^* = \sqrt{\sigma^2/(\kappa A)}$ (Eq. 10) into the sparse error estimate $\mathbb{E}[\delta_i^2]_{\text{sparse}} = \sigma^2/k + \kappa(kA + B) + \text{lag}^2$ (Eq. 8), the exploration term and the self-noise term become equal, giving:

$$\mathbb{E}[\delta_i^2]_{\text{sparse}}(k^*) \;=\; 2\,\sigma\sqrt{\kappa A} \;+\; \kappa B \;+\; \text{lag}^2 \tag{11}$$

Comparing against dense exploration ($\sigma^2 + \kappa(A + B) + \text{lag}^2$), the coupling κB and the lag cancel, and the reduction in effective squared miscalibration collapses to a perfect square (using $\sqrt{\kappa A} = \sigma/k^*$):

$$\Delta\,\mathbb{E}[\delta_i^2] \;=\; \sigma^2 + \kappa A - 2\sigma\sqrt{\kappa A} \;=\; \left(\sigma - \sqrt{\kappa A}\right)^2 \;=\; \sigma^2\Big(1 - \tfrac{1}{k^*}\Big)^2 \tag{12}$$

Through the power law (Section 4.1) this is an LER saving $\Delta p_L/p_L \approx \frac{d+1}{2}\,P\Omega\,(\sigma - \sqrt{\kappa A})^2/\tilde{\varepsilon}$. Comparing this against the dense exploration gap of $\frac{d+1}{2}\,P\Omega\sigma^2/\tilde{\varepsilon}$ (Eq. 2); the optimal saving above is exactly a fraction of it:

$$\frac{\Delta p_L^{\text{saving}}}{\Delta p_L^{\text{gap}}} \;=\; \Big(1 - \tfrac{1}{k^*}\Big)^2 \;=\; \Big(1 - \sqrt{\tfrac{\kappa A}{\sigma^2}}\Big)^2 \tag{13}$$

It depends only on the single ratio $\sigma^2/(\kappa A) = k^{*2}$: how strongly the exploration noise σ^2 dominates the per-parameter gradient noise κA: at $k^* = 1$ the saving is zero (dense is already optimal), and as $\kappa A/\sigma^2 \to 0$ the gap is removed entirely.

5.2 Adaptive sparsity

The optimal value of k for sparse exploration derived in Section 5.1 (Eq. 10), is dependent on factors that differ per parameter. It is dependent on $\bar{\delta}_i$ and σ_i, which differ per epoch. This motivates choosing a different value of $\hat{k}_i$ per parameter and per epoch. The estimator $\hat{k}_i$ is determined from information available to the controller at runtime, as derived below.

Both runtime estimators below are built from the controller's smoothed gradient $\hat{g}_i$. Since $\hat{g}_i^2$ overestimates the squared mean gradient $(\mathbb{E}[\nabla_{\mu_i} J])^2$ by its own variance $\text{Var}(\hat{g}_i)$, we de-bias it with Adam's second moment $\hat{v}_i$ (derivation in Appendix C.1) and use Eq. 14 in both estimators.

$$\hat{g}_{i,\text{db}}^2 = \max\Big(\hat{g}_i^2 - \tfrac{1-\beta_1}{2\beta_1}\,\max(\hat{v}_i - \hat{g}_i^2,\,0),\;0\Big) \tag{14}$$

Now to find the estimator for the optimal $\hat{k}_i$: Two effects make this optimum differ from the constant-κ form of Section 5.1. First, the Adam optimizer shrinks its effective step $\eta_{\text{eff}} = \text{lr}/\sqrt{\hat{v}}$ when the gradient is noisier (with $\hat{v}$ its second-moment estimate), so the noise gain κ is itself k-dependent. Second, a drifting optimum leaves the policy mean with a tracking lag. Treating the controller as a feedback loop around the drifting optimum folds both into a single effective gain (Appendix C.2):

$$k^* = \sqrt{\frac{\sigma^2}{\kappa_{\text{eff}}\, A}}, \qquad \kappa_{\text{eff}} = \tfrac{1}{2}\kappa + \kappa_{\text{lag}}, \qquad \kappa = \frac{\eta_{\text{eff}}}{2SM_{1/2}}, \quad \kappa_{\text{lag}} = \frac{\hat{g}_{\text{db}}^2}{\hat{v}\,S^2 M_{1/2}} \tag{15}$$

where $S = 2\,n_{re}\,c_r\,\Omega$ is the local restoring strength of the gradient. This is exactly Eq. 10 with $\kappa \to \kappa_{\text{eff}}$, so every result of Section 5.1 carries over.

This optimum still depends on the per-parameter quantity A, which the controller cannot use directly: A depends on $\bar{\delta}_i = \mu_i - p_i^{opt}$, and p_i^{opt} is unknown to the controller.

$$\begin{aligned}\mathbb{E}[A] = &\quad \mathbb{E}\left[8\, n_{re,i}^2\, c_{r,i}^2 \Omega_i^2 \bar{\delta}_i^2 + n_{re,i}\, s^2/(2\sigma_i^2)\right] \\ = &\quad 2\,\mathbb{E}\left[\left(2\, n_{re,i}\, c_{r,i}\, \Omega_i\, \bar{\delta}_i\right)^2\right] + n_{re,i}\, s^2/(2\sigma_i^2) \end{aligned} \tag{16}$$

Here s^2 is not a random variable but a fixed property of the measurement (the variance of the measurement noise), so it can be taken outside the expectation. The quantity being squared in Eq. 16 is exactly the expected gradient found in Appendix B, $\mathbb{E}[\nabla_{\mu_i} J] \approx 2\, n_{re,i}\, c_{r,i}\, \Omega_i\, \bar{\delta}_i$, evaluated at the current miscalibration. In other words, the signal part of $\mathbb{E}[A]$ is simply twice the squared expected gradient:

$$\mathbb{E}[A] \;=\; 2\,\mathbb{E}\left[\left(\mathbb{E}[\nabla_{\mu_i} J]\right)^2\right] + n_{re,i}\, s^2/(2\sigma_i^2) \tag{17}$$

This removes the dependence on $\bar{\delta}_i$: the squared expected gradient $(\mathbb{E}[\nabla_{\mu_i} J])^2$ is exactly what the debiased signal $\hat{g}_{i,\text{db}}^2$ estimates, so $\mathbb{E}[A]$ can be estimated at runtime as:

$$\hat{A}_i \;=\; 2\,\hat{g}_{i,\text{db}}^2 + n_{re,i}\, s^2/(2\sigma_i^2) \tag{18}$$

where $n_{re,i}$ is fixed by the circuit layout and s^2 is measured once before running the controller.[11]

Substituting $\hat{A}_i$ into Eq. 15 gives the final per-parameter, per-epoch estimator:

$$\hat{k}_i \;=\; \text{clip}\left(\sqrt{\frac{\sigma_i^2}{\kappa_{\text{eff}}\, \hat{A}_i}},\; 1,\; M_{1/2}\right) \tag{19}$$

This value of $\hat{k}_i$ is clipped so that every parameter is sampled at least once and at most $M_{1/2}$ times ($1 \le \hat{k}_i \le M_{1/2}$). To maintain an unbiased $\hat{k}_i$, we round the number of selected samples $M_{1/2}/k$ stochastically: we use $\lfloor M_{1/2}/k \rfloor$ samples plus one extra if $M_{1/2}/k - \lfloor M_{1/2}/k \rfloor < r$, with r randomly chosen in $[0, 1]$.

5.3 Empirical confirmation

The two estimator ingredients (the measure-term coefficient n_{re} and the gradient-signal de-bias) are validated in Appendix C.3. Figure 3a validates Eq. 5, proving the downside of sparse sampling is less important for higher values of P. Figure 3b shows the optimal value of k shrinks for fast drift speeds. This was tested by evaluating the solution on a linear drift with a fixed drift speed, then evaluating the LER of the last few cycles. Some of these LER traces can be found in Appendix G (Fig. 13), from which it can be seen that for fast drift speeds the solution fails to track the optimum (gray squares). The adaptive $\hat{k}$ estimate mostly tracks this valley and always gives a close to optimal result, without any further tuning.

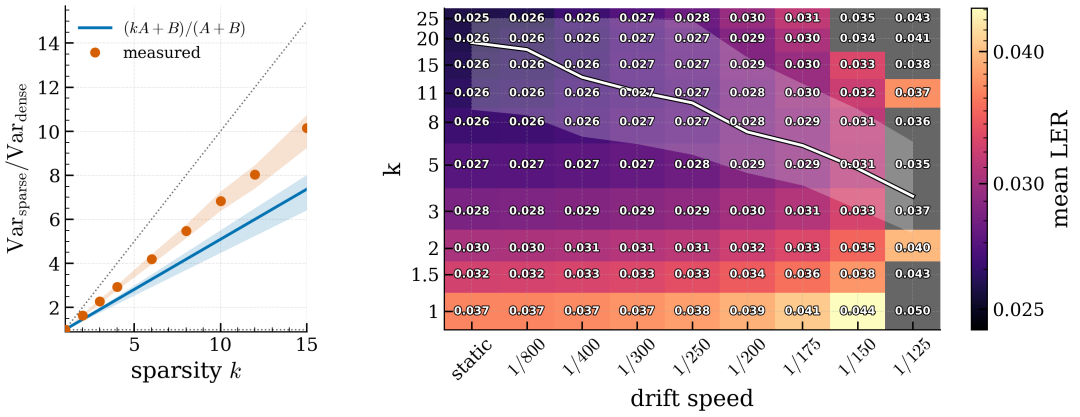
6 Results

The sparse solution (fixed-k and adaptive) is evaluated across drift speeds, code distances, and parameter counts. Unless stated otherwise, experiments are run using $d = 3$ and $P = 6$, with the latter being based on the number of parameters per gate (5–7) used on the real quantum computer by Sivak et al. [1]. Exact configuration of hyperparameters can be found in Appendix H or the source code.

6.1 Behavior under drift

In order to estimate the performance of sparse sampling under real drift, the solution is tested against a few different kinds of noise. First, solutions are tested against a sinusoidal drift (Figure 4a), similar to

[11] The value of s^2 can be determined by performing a number of measurements at a fixed policy and then finding the variance of the rewards.



(a) Gradient-variance ratio vs. k.

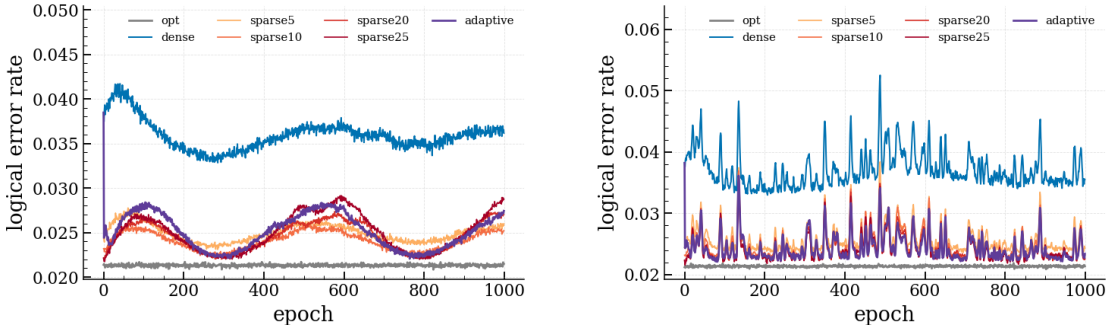
(b) Steady-state LER over k and drift speed; adaptive $\hat{k}$ overlaid.

Figure 3: Empirical confirmation of the Section 5.1 predictions ($d = 3$, $P = 6$). **(a)** The measured gradient-variance ratio (orange) against sparsity k tracks the prediction $(kA+B)/(A+B)$ (Eq. 5, blue) and stays well below the naive ratio $= k$ (dotted), so the variance grows far slower than k. **(b)** Steady-state LER over sparsity k and drift speed; the adaptive estimator's $\hat{k}$ (white line is median $\hat{k}$ estimate, band is [q1, q3]) tracks the per-speed optimum (the LER valley in each column); for the grayed-out cells, the steering is unable to track the drift and the LER keeps increasing. Runs are evaluated at $P = 6$ (based on real amounts used by Sivak et al.; see Section 6).

the one used by Sivak et al. Under this drift, the sparse policies perform similarly (Table 1): $k = 10$ is marginally lowest at mean LER $= 0.0241$, with $k = 20$ next (0.0246) and $k = 5$, $k = 25$, and adaptive k slightly higher at 0.0250, 0.0250, and 0.0251. All sparse solutions still outperform dense exploration by a large margin (dense has mean LER $= 0.0359$). The solutions reduce the exploration gap by roughly 74–81%, with the gap compared to the maximum theoretical reduction being attributed to the solutions being unable to track the noise at high speeds.[12]

Next, the solutions are tested against a hardware-inspired band-limited $1/f$ drift (Figure 4b), whose frequency content is taken from device measurements of Sivak et al. Only this frequency content is matched to the device, not the amplitude, phase, or cross-parameter structure, so the drift is a robustness check rather than a faithful device model (Appendix F). Here dense exploration again sits well above the sparse solutions, with mean LER $= 0.0362$ against an optimal-policy floor of 0.0213. All sparse

[12]Unlike Figure 2b, where similar drift was successfully tracked, here $P = 6$ was used, making the tracking worse.



(a) Sinusoidal drift, one full period over the run.

(b) Hardware-inspired band-limited $1/f$ drift (Appendix F).

Figure 4: Per-epoch LER under continuous drift ($d = 3$, $P = 6$) for the optimal policy, dense exploration ($k = 1$), sparse exploration at several k, and adaptive k. In both cases the sparse policies sit well below dense, between it and the optimal-policy floor.

Table 1: Per-cycle LER by policy across the drift scenarios ($d = 3$, $P = 6$). For the two continuous drifts, "red." is the reduction of the exploration gap (the mean LER above the optimal policy) relative to dense, in percent. For the step response, "peak" is the post-step maximum LER, "ep. 100" the LER 50 epochs after the step, and "steady" the mean LER over the last 50 epochs.

	Sinusoidal		Hardware-inspired $1/f$		Step response		
Policy	LER	red. %	LER	red. %	peak	ep. 100	steady
optimal	0.0213	100	0.0213	100	0.0246	0.0242	0.0242
dense	0.0359	0	0.0362	0	0.1443	0.0779	0.0374
sparse-5	0.0250	75	0.0251	75	0.1264	0.0604	0.0271
sparse-10	0.0241	81	0.0240	83	0.1221	0.0609	0.0262
sparse-20	0.0246	77	0.0235	85	0.1206	0.0642	0.0266
sparse-25	0.0250	75	0.0236	85	0.1239	0.0653	0.0273
adaptive	0.0251	74	0.0237	85	0.1252	0.0776	0.0275

solutions land in a narrow band just above this floor (Table 1): the sparsest settings $k = 20$ and $k = 25$ are marginally lowest (mean LER $= 0.0235$ and 0.0236), with adaptive k (0.0237) and $k = 10$ (0.0240) close behind, and $k = 5$ slightly higher (0.0251). Every tested sparse policy closes between roughly 75% and 85% of the exploration gap under this hardware-inspired band-limited drift. Unlike the fast sinusoidal excursion, this slow $1/f$ drift is tracked well at $P = 6$, so the remaining gap is mostly the residual exploration cost rather than a tracking limit.

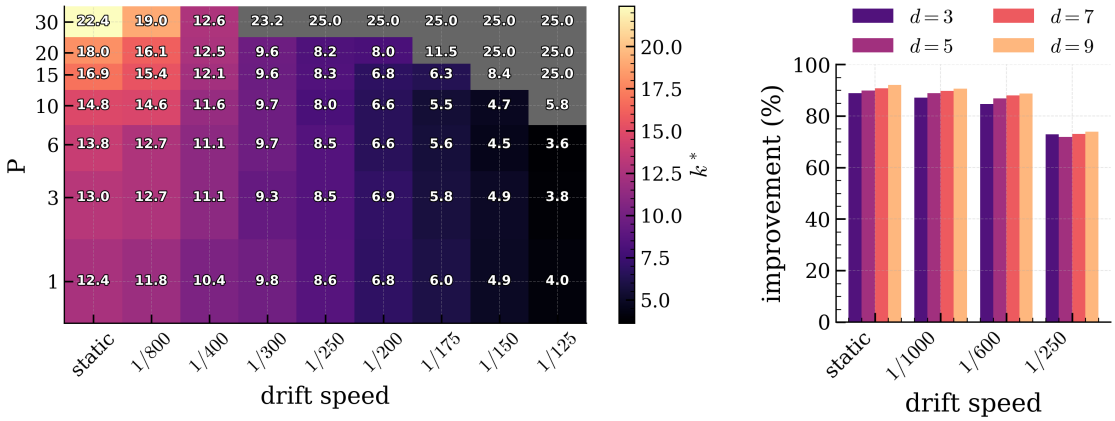
Finally, the solutions are tested against a sudden step in the optimal parameters (full plot in Appendix G, Figure 14). The fixed-k sparse policies match or beat dense throughout the recovery: dense and the adaptive controller re-converge slightly faster, but that speed does not outweigh their higher error rate during the transient. Every policy spikes at the step (epoch 50) and then recovers. The fixed-k sparse policies spike lower (peak LER about 0.12 against dense's 0.144), are already down to around 0.06 by epoch 100 (against dense's 0.078), and settle to 0.026–0.027 against dense's 0.037 (Table 1). The adaptive controller behaves differently: it spikes low and settles low, like the fixed-k policies, but at epoch 100 it sits at 0.078, around dense's level. The reason is that the adaptive estimator is derived to minimize the steady-state LER (Section 5.2), not to find the optimal k while the learned policy is still converging: after the spike, $\hat{k}$ shrinks to a low value to recover the learned policy faster, but for this small step the sparse solutions, which stay at high k, still win despite their slightly slower convergence.

The steady-state behavior the adaptive k was designed for does hold up (Figure 3b), so the problem is that the current adaptive solution does not do well when there is no steady-state; this should be a focus of future work (Section 7).

6.2 Scaling: code distance and control dimension

In order to show how the optimal k scales with different numbers of parameters per gate P, a grid search was performed on drift speed and P (Figure 5a). All solutions were evaluated using linearly increasing p^{opt} at a fixed speed. The value plotted is the adaptive k estimate, which is accurate for the optimal k for steady-state performance. As the analysis of Section 5.1 predicts, the chosen k rises with the number of parameters per gate P and falls as the drift speeds up: more parameters give more error to save by perturbing fewer at a time, while faster drift demands denser sampling to keep tracking. For high drift speeds with many per-gate parameters, the solution is unable to track the optimum (the LER just keeps climbing), so those runs measure nothing meaningful and are discarded, as can be seen in the LER and k traces found in Appendix G (Figure 15, Figure 16).

In order to evaluate how the solution scales with distance, the solution was evaluated at a few drift speeds for distances $3, 5, 7, 9$ (Figure 5b). The solutions were again evaluated using linear drift to get steady-state performance. As expected, the percentage reduction in exploration gap is roughly independent of code distance over the tested range ($d = 3$ to 9), sitting near the maximum theoretical reduction of roughly 90% found in Section 5.1. A sweep over the irreducible error rate $\tilde{\varepsilon}$ shows the same independence



(a) Adaptive k^* versus control parameters per gate P and drift speed. Grayed-out cells are discarded, marking where steering can no longer track the optimum (Appendix G).

(b) Improvement of adaptive sparse over dense across various code distances and drift speeds.

Figure 5: Scaling of the sparse-adaptive solution with control dimension P (a) and code distance d (b). The chosen sparsity k^* rises with P and falls as the drift speeds up (a), while the gap reduction stays near its theoretical ceiling across distances (b).

there: the reduction stays near 90% across the swept range (Appendix G, Figure 17). The slightly worse performance of lower code distances can likely be explained by the higher ratio of gates on the edge, which have fewer surrounding parameters and therefore perform worse (Section 5.1). The performance only worsens when the solution becomes unable to properly track the drift at high drift speeds, in which case it needs to go to a lower value of k.

7 Discussion

This section interprets the results: what sparse exploration gains, where it falls short, and what changes on real hardware. As found in Section 4, the relative exploration gap grows with code distance $(d+1)$ and falling irreducible error rate $(1/\tilde{\varepsilon})$, and compounds over the run length. This makes it a growing share of the error rate as hardware scales to useful sizes. Section 6 shows that sparse exploration reduces this gap across the regimes a real device is likely to encounter. Under both the sinusoidal drift of Sivak et al. [1] and a hardware-inspired band-limited $1/f$ drift, the tested sparse policies close roughly 74–85% of the exploration gap relative to dense steering (Table 1), and the optimal sparsity follows the trend predicted in Section 5.1: the chosen k grows with the number of control parameters per gate P and shrinks as the drift speeds up (Figure 5a). This relative reduction is essentially independent of code distance over the tested range ($d \leq 9$) and sits near its theoretical ceiling of roughly 90% for static-to-moderate drift (Figure 5b). Because the exploration gap itself grows with distance, the absolute benefit is largest on the large-distance codes needed for useful computation.

The cost of the solution is also minimal: even under sudden steps in the optimum p, the sparse solutions perform comparably to or better than dense exploration. The one exception is the adaptive solution derived in Section 5.2: although it performs well under steady drift, empirical tests show it performs worse than a fixed-k sparse solution under fast or sudden drift. This is to be expected, since the estimated $\hat{k}$ is based on steady-state performance.

7.1 Replicability on real hardware

The simulation is a deliberate simplification of real quantum computers, used for manageable runtimes. The Stim [17] surface code circuit is based on a Stim model of a hardware-derived XZZX surface code with SI1000 noise [21] to simulate errors, which is a commonly used, hardware-inspired circuit-level

noise model. Our model has been verified to be comparable to existing implementations (Section 3.2). The main simplification is the relation between the parameters and errors used.

This paper mainly focuses on the reinforcement learning model, its convergence, and its cost. This model is accurately replicated from Sivak et al. The results only lean on the fact that miscalibration of the RL model causes an elevated number of error detection events [1] and that miscalibrating only a part of the qubits in turn gives a lower LER. The latter assumption leans on the physical structure of the surface code and how logical errors are formed [20].

Additionally, some further assumptions limit the realism of the simulation, such as: only depolarizing errors are used,[13] the abstraction of parameters to P identical parameters per gate [1], the drift being shared across all gates with a single timescale, and the hardware-inspired drift model only being assumed semi-accurate.[14] Note that sparse exploration keeps more parameters at their calibrated values at once, which may modestly reduce parameter-dependent crosstalk; this is a plausible side benefit we do not test, since the Stim simulation does not accurately model crosstalk.

Finally, the hardware implementation of Sivak et al. already had some limitations: it forces the controller to run slower than the theoretically best speed, and introduces some control latency. The latter adds to the lag term, where sparse exploration is already weakest, so it could shrink the margin of improvement. These assumptions suggest that the exploration/tracking mechanism may carry over, but the quantitative margin of improvement can only be confirmed on hardware.

Several directions remain open. The most direct is validation on a real quantum computer, the only way to confirm that the simplifications of the simulation carry over. The adaptive estimator is currently derived for steady-state drift and reacts suboptimally to sudden steps, so deriving a variant that also accounts for this is a natural extension, though it would only be meaningful when tuned to a device's measured drift, as the current hardware-inspired estimate is not guaranteed to be accurate (Appendix F). Finally, hardware testing at higher code distances and over the multi-day program lengths of practical algorithms would establish whether the exploration gap, and its reduction, behave as predicted at scale.

8 Conclusion

In this paper we replicated the simulation and PGPE-based continuous calibration of Sivak et al. [1], reproducing their key published results (Figure 2), and introduced sparse exploring, a random-masking variant that perturbs only about $1/k$ of the control parameters in each sample in order to reduce the exploration gap. We showed both theoretically (Section 5.1) and empirically (Section 6) that the lower mean error rate during exploration outweighs the slightly increased gradient variance and tracking lag, and we derived an estimator that selects the sparsity k online.

Across both sinusoidal and hardware-inspired band-limited $1/f$ drift, every tested sparse policy closes roughly 74 to 85% of the exploration gap relative to dense steering, reaching up to roughly 90% for slow drift, essentially independent of code distance over the tested range ($d = 3$ to 9). Since the exploration gap grows with code distance and falling irreducible error rate, and compounds over program length, this benefit is largest on the large codes that useful computation will require. The online estimator for the optimal sparsity performs well for a steady-state learned policy and slow drift, but does not account for changing drift speeds or fast-moving optima. In these cases, although it still outperforms dense exploration, sparse masking with a fixed k gives better results.

Overall, sparse exploring is a cheap addition to online steering that lowers the error rate over the multi-day programs useful algorithms will require, with the largest gains on the large-distance codes.

[13] Equal likelihood of X, Z and Y errors; no leakage, crosstalk, cosmic-ray or non-Markovian effects [4].

[14] It shares similar frequencies with real drift, which is the main property being tested; see Appendix F.

9 Responsible Research

All code used in this paper is made available publicly under the Apache 2.0 license.[15] Replication uses Python 3.11, Stim 1.13 [17], PyMatching 2.3.1 [18, 19], and NumPy 2.4.4. The replication is based on version 3 (March 7th, 2026) of Sivak et al. [1], which is the latest version at the time of writing. All findings use a single fixed seed. Each reported value is averaged over many QEC cycles within a run, which suppresses measurement noise but does not capture seed-to-seed variation; the close rankings between adjacent sparsity settings should be read with this in mind. Hyperparameters used can be found in the repository, as well as in Appendix H. Code for all plots and experiments is also located in the repository, with most having a manageable runtime due to the usage of multiprocessing.

The replication of Sivak et al. is as faithful as possible, with most missing details being related to the exact configuration of the PGPE controller. Assumptions in the replication are discussed in Section 3. Proper tests and comparisons have been used to verify the behavior of the replication matches the original as closely as possible. The only data used are the Stim files provided by Sivak et al., to verify the correctness of the XZZX surface code implementation. This data is made publicly available with the paper.

The findings presented are verified with both theoretical calculations and empirical evidence. Assumptions made for these claims are discussed in Section 5 and Section 7. As mentioned, the claimed results have been verified with theory and simulation, and still need to be verified on actual quantum computing hardware.

This work contributes to the broader effort toward fault-tolerant quantum computing, which will eventually be able to break widely used cryptography such as RSA [3]. This is a real long-term security concern, but it is being addressed at the field level through the ongoing transition to post-quantum cryptography.

Large language models (primarily Claude, Anthropic) were used as a coding and writing aid: debugging the simulation, writing plotting code, generating unit tests, checking spelling and grammar, suggesting phrasing, and cross-checking the algebra in the derivations of Section 5 and the appendices. They were not used to generate experimental data or results. All such output was reviewed, and the derivations and code were independently verified against simulation; the author takes full responsibility for the content of this paper.

Acknowledgements

I thank my supervisors, Rihan Hai and Tim Littau, for their guidance throughout this project, and Tim Littau and Maximiliaan van der Veek for several rounds of feedback on the paper.

[15] https://github.com/InfiniData-Lab/qec-sparse-exploration

References

[1] Google Quantum AI and Google DeepMind. Reinforcement learning control of quantum error correction, 2026. arXiv:2511.08493v3.

[2] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, UK, 10th anniversary edition, 2010.

[3] Craig Gidney. How to factor 2048 bit RSA integers with less than a million noisy qubits, 2025.

[4] Google Quantum AI, Rajeev Acharya, et al. Quantum error correction below the surface code threshold. *Nature*, 638:920–926, 2025.

[5] P. V. Klimov, J. Kelly, Z. Chen, M. Neeley, A. Megrant, B. Burkett, R. Barends, K. Arya, B. Chiaro, Yu Chen, A. Dunsworth, A. Fowler, B. Foxen, C. Gidney, M. Giustina, R. Graff, T. Huang, E. Jeffrey, Erik Lucero, J. Y. Mutus, O. Naaman, C. Neill, C. Quintana, P. Roushan, Daniel Sank, A. Vainsencher, J. Wenner, T. C. White, S. Boixo, R. Babbush, V. N. Smelyanskiy, H. Neven, and John M. Martinis. Fluctuations of energy-relaxation times in superconducting qubits. *Physical Review Letters*, 121(9):090502, August 2018.

[6] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[7] James C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[8] Albert S. Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics*, 22(2):507–560, 2022.

[9] HanQin Cai, Yuchen Lou, Daniel McKenzie, and Wotao Yin. A zeroth-order block coordinate descent algorithm for huge-scale black-box optimization. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 1193–1203, 2021.

[10] Mayumi Ohta, Nathaniel Berger, Artem Sokolov, and Stefan Riezler. Sparse perturbations for improved convergence in stochastic zeroth-order optimization. In *Machine Learning, Optimization, and Data Science (LOD 2020)*, volume 12566 of *Lecture Notes in Computer Science*, pages 39–64, Cham, 2020. Springer.

[11] Krzysztof Choromanski, Mark Rowland, Vikas Sindhwani, Richard E. Turner, and Adrian Weller. Structured evolution with compact architectures for scalable policy optimization. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 970–978, 2018.

[12] Bernard Widrow, John M. McCool, Michael G. Larimore, and C. Richard Johnson, Jr. Stationary and nonstationary learning characteristics of the LMS adaptive filter. *Proceedings of the IEEE*, 64(8):1151–1162, 1976.

[13] Volodymyr Sivak, Michael Newman, and Paul Klimov. Optimization of decoder priors for accurate quantum error correction. *Physical Review Letters*, 133(15):150603, 2024.

[14] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2), June 2019.

[15] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[16] Matt McEwen, Dave Bacon, and Craig Gidney. Relaxing hardware requirements for surface code circuits using time-dynamics. *Quantum*, 7:1172, 2023.

[17] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021.

[18] Oscar Higgott. PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching, 2021.

[19] Oscar Higgott and Craig Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *Quantum*, 9:1600, 2025.

[20] J. Pablo Bonilla Ataides, David K. Tuckett, Stephen D. Bartlett, Steven T. Flammia, and Benjamin J. Brown. The XZZX surface code. *Nature Communications*, 12(1):2172, 2021.

[21] Craig Gidney, Michael Newman, Austin Fowler, and Michael Broughton. A fault-tolerant honeycomb memory. *Quantum*, 5:605, 2021.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[23] Ronald J. Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[24] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[25] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

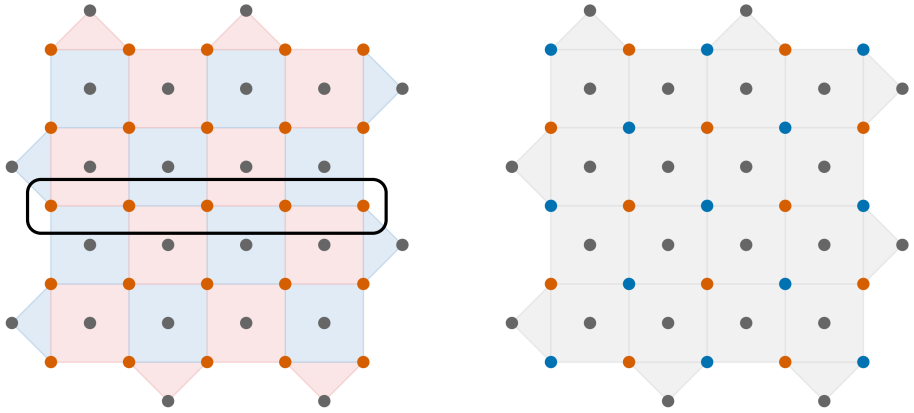
[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2015.

A Quantum error correction and the surface code

Similar to how a classical computer does computations using bits, a quantum computer makes use of qubits. A qubit is a physical system with two distinguishable quantum states, conventionally labeled $|0\rangle$ and $|1\rangle$. Unlike classical bits, qubits can be placed in superpositions of $|0\rangle$ and $|1\rangle$. This means the physical system is in a quantum superposition between the two designated states. The superposition is defined by two properties: first the relative weights of the two base states $|0\rangle$ and $|1\rangle$, which determine the probability of the qubit being measured in the respective states; and second, the relative phase between them, which does not influence the measured value of the qubit, but can be measured by first applying gates. Measuring, and thus observing, a qubit will collapse this superposition into a definite classical outcome (the probability of measuring a qubit in a given base state is the squared relative weight of that base state). A measurement can be made in the Z basis (distinguishing $|0\rangle$ and $|1\rangle$) or the X basis ($|+\rangle$, $|-\rangle$); the surface code uses both, since X-type errors are caught by Z measurements and Z-type errors by X measurements. Two qubits can be *entangled* so that their measurement outcomes are correlated; for example, in the Bell state $(|00\rangle + |11\rangle)/\sqrt{2}$, measuring one qubit as $|0\rangle$ forces the other to $|0\rangle$ as well, and this correlation holds regardless of the distance between them [2, Ch. 1]. One consequence of these properties of qubits is the inability to copy information from one qubit to another, as stated in the no-cloning theorem [2, Ch. 10]. It is, however, possible to entangle the qubits so they share the same measurement outcome; but since measuring one of them collapses the state of all entangled qubits, this makes quantum error correction difficult.

Quantum error correction (QEC) schemes have some similarity to classical error correction schemes: they store information redundantly across multiple physical qubits. But rather than measuring the value of each qubit, it is possible to measure only the parity of the values between two qubits without collapsing the encoded state (and similarly for phases) [2, Ch. 10]. Although qubits can have continuous errors, these parity measurements project the error onto one of a discrete set of Pauli errors, which can then be detected and corrected with a few operations. This fact makes QEC feasible where analog error correction is not.

One such QEC scheme is the rotated surface code, which can be seen in Figure 6a. In a surface code of distance d, a grid of $d \times d$ physical data qubits is used to redundantly store the value of a single "logical" qubit. Additionally, $d^2 - 1$ ancilla qubits are positioned between the data qubits to perform measurements. The ancilla qubits in the standard surface code alternate between X- and Z-type measurements (referred to as stabilizer measurements), correcting Z and X basis errors respectively. These detection zones are signified by the blue and red areas in Figure 6a. A logical operator in the Z basis can be performed on the logical qubit by applying the gate to all qubits in a chain spanning from the left to the right border of the surface code. Similarly, operators in the X basis can be performed by spanning a chain of operations from the top to the bottom. The cheapest way of performing either is by applying the operations to any single row or column of data qubits, respectively (a weight-d representative). A logical error occurs when such a chain is formed by errors on individual physical qubits. A chain of length d requires d physical errors to form, but the decoder may already misidentify the error pattern once roughly $(d + 1)/2$ errors have occurred. The surface code's error-correcting capability therefore improves with the distance d.



(a) Standard surface code.

(b) XZZX surface code.

Figure 6: Distance $d = 5$ surface code variants. **(a)** Standard rotated surface code. Red qubits are data qubits, gray qubits are ancilla qubits. Blue areas mark X-type stabilizers; red areas mark Z-type stabilizers. A row of data qubits is highlighted as a representative logical operator. **(b)** XZZX variant used by Sivak et al. and in this work.

A variation of the rotated surface code, used by Sivak et al. and in this paper, is the XZZX surface code (Figure 6b). Here, the data qubits alternate between being in the Z and X bases. As a result, instead of alternating X- and Z-type checks as in the standard variant (Figure 6a), every ancilla measures the same mixed XZZX stabilizer: a product of X and Z Paulis on its neighboring data qubits, so one stabilizer type catches both X- and Z-basis errors. The code distance d carries over unchanged. On real hardware, this variant is less susceptible to errors in environments where either X or Z type errors dominate [20]; in this work, the XZZX variant is used purely to match the implementation of Sivak et al.

B Gradient variance derivation

This appendix derives a formula for the variance of the gradient for the PGPE controller. This derivation is done under the error model of Section 4.1 with symmetric sampling [6]. The further optimizations to the PGPE controller (PPO importance clipping, gradient clipping, the replay buffer) will either keep the variance the same or reduce it, never increase it.[16] In order to keep the calculation manageable, these factors are omitted, and the current estimation of the variance is taken as an upper bound.

In a symmetric pair, every perturbed parameter uses parameters $p_i^+ = \mu_i + \gamma_i$ and $p_i^- = \mu_i - \gamma_i$, where $\gamma_i \sim \mathcal{N}(0, \sigma_i^2)$. Write $\bar{\delta}_i = \mu_i - p_i^{opt}$ for the miscalibration of the policy mean of parameter i, so that $\delta_i^\pm = p_i^\pm - p_i^{opt} = \bar{\delta}_i \pm \gamma_i$. Note the bar: $\bar{\delta}$ is the miscalibration of the learned mean, while δ without a bar (Section 4.1) is the miscalibration of an applied parameter; the two differ by the applied perturbation. Let the influence of this pair on the gradient $\nabla_{\mu_i} J(p)$ be defined as g_i, which is then approximately [6]:

$$g_i(p) \approx \frac{\gamma_i (r^+ - r^-)}{2\sigma_i^2} \tag{20}$$

where $r^\pm$ is the reward from $p_i^\pm$. Parameter i is observed by $n_{re,i}$ detectors, each of which contributes to the reward. Since $\varepsilon_i^\pm$ has a linear relation with the reward, the reward consists of three contributions: a term from parameter i itself, terms from the other perturbed parameters that share a detector with i, and the measurement noise. Because these enter $r^+ - r^-$ additively, their effects on g_i can be analyzed separately and added together at the end. We start with the contribution of parameter i itself, $r^\pm \approx n_{re,i}\, c_{r,i}\, \Omega_i\, (\delta_i^\pm)^2$, where $c_{r,i}$ is the coefficient between error rate and reward (linear by approximation, with dropped cross-terms being only $\mathcal{O}(\text{PER}^2)$). Using this we find:

$$\begin{aligned} g_i(p) \approx & \frac{n_{re,i}\, c_{r,i}\, \gamma_i\, \Omega_i ((\bar{\delta}_i + \gamma_i)^2 - (\bar{\delta}_i - \gamma_i)^2)}{2\sigma_i^2} \\ = & \frac{4\, n_{re,i}\, c_{r,i}\, \gamma_i^2\, \Omega_i \bar{\delta}_i}{2\sigma_i^2} = \frac{2\, n_{re,i}\, c_{r,i}\, \gamma_i^2\, \Omega_i \bar{\delta}_i}{\sigma_i^2} \end{aligned} \tag{21}$$

Note that since $\gamma_i \sim \mathcal{N}(0, \sigma_i^2)$, this estimator for $g_i(p)$ has mean $2\, n_{re,i}\, c_{r,i}\, \Omega_i\, \bar{\delta}_i$, which is the true gradient of the pooled reward; so the estimator is unbiased. We now find:

$$\begin{aligned} \text{Var}\left(g_i(p)\right) \approx & \text{Var}\left(\frac{2\, n_{re,i}\, c_{r,i}\, \gamma_i^2\, \Omega_i \bar{\delta}_i}{\sigma_i^2}\right) \\ = & \mathbb{E}\left[\left(\frac{2\, n_{re,i}\, c_{r,i}\, \gamma_i^2\, \Omega_i \bar{\delta}_i}{\sigma_i^2}\right)^2\right] - \mathbb{E}\left[\frac{2\, n_{re,i}\, c_{r,i}\, \gamma_i^2\, \Omega_i \bar{\delta}_i}{\sigma_i^2}\right]^2 \\ = & \frac{4\, n_{re,i}^2\, c_{r,i}^2\, \mathbb{E}\left[\gamma_i^4\right]\, \Omega_i^2 \bar{\delta}_i^2}{\sigma_i^4} - \frac{4\, n_{re,i}^2\, c_{r,i}^2\, \mathbb{E}\left[\gamma_i^2\right]^2\, \Omega_i^2 \bar{\delta}_i^2}{\sigma_i^4} \\ = & 8\, n_{re,i}^2\, c_{r,i}^2\, \Omega_i^2 \bar{\delta}_i^2 \end{aligned} \tag{22}$$

Additionally, each of the $n_{re,i}$ detector measurements carries an independent sampling-noise term with mean 0 and variance s^2; write $s^\pm$ for their sum, which has mean 0 and variance $n_{re,i}\, s^2$. These terms add variance:

$$\text{Var}\left(\frac{\gamma_i (s^+ - s^-)}{2\sigma_i^2}\right) = \mathbb{E}\left[\left(\frac{\gamma_i (s^+ - s^-)}{2\sigma_i^2}\right)^2\right] - \mathbb{E}\left[\frac{\gamma_i (s^+ - s^-)}{2\sigma_i^2}\right]^2 = \frac{n_{re,i}\, s^2}{2\sigma_i^2} \tag{23}$$

So far the reward only contained parameter i. In the full reward, every other perturbed parameter l that shares a detector with i adds its own term $4\, c_{r,l}\, \Omega_l\, \bar{\delta}_l\, \gamma_l$ to $r^+ - r^-$, by exactly the expansion above.

[16] Clipping does introduce a small bias, but this can be neglected.

Through the $\gamma_i/(2\sigma_i^2)$ factor of the estimator, this becomes a term $2\,c_{r,l}\,\Omega_l\,\bar{\delta}_l\,\gamma_i\gamma_l/\sigma_i^2$ in g_i. Because γ_i and γ_l are independent and zero-mean, this term has mean 0 (so the estimator stays unbiased), and its variance follows the same computation as for the sampling noise: $(2c_{r,l}\Omega_l\bar{\delta}_l/\sigma_i^2)^2 \cdot \mathbb{E}[\gamma_i^2]\,\mathbb{E}[\gamma_l^2] = 4\,c_{r,l}^2\Omega_l^2\bar{\delta}_l^2 \;\approx\; 4\,c_r^2\Omega^2\bar{\delta}^2$ (using $\sigma_l = \sigma_i$, per the equal-σ assumption of Section 4.1). With n_{co} such parameters,[17] all uncorrelated, a single pair has variance:

$$8\,n_{re,i}^2\,c_{r,i}^2\Omega_i^2\bar{\delta}_i^2 + 4\,n_{co}\,c_r^2\Omega^2\bar{\delta}^2 + n_{re,i}\,s^2/(2\sigma_i^2) \tag{24}$$

The expected value of the gradient for a single pair contains only parameter i's own term: the measurement noise and all coupling terms have expected value 0, so the expected value stays equal to the true gradient. An epoch contains $M_{1/2}$ pairs in which parameter i is perturbed: $M_{1/2} = M/2$ for normal exploration. The estimates from different pairs are independent and identically distributed, so the batch gradient (their average) has the same mean and $1/M_{1/2}$ times the variance:

$$\mathbb{E}\left[\nabla_{\mu_i} J\right] \approx 2\,n_{re,i}\,c_{r,i}\,\Omega_i\,\bar{\delta}_i \tag{25}$$

$$\mathrm{Var}(\nabla_{\mu_i} J) \approx \frac{1}{M_{1/2}}\,\left(8\,n_{re,i}^2\,c_{r,i}^2\Omega_i^2\bar{\delta}_i^2 + 4\,n_{co}\,c_r^2\Omega^2\bar{\delta}^2 + n_{re,i}\,s^2/(2\sigma_i^2)\right) \tag{26}$$

This batch average is the gradient estimate the controller computes in practice. Smoothing it once more across epochs yields the estimate $\hat{g}_i$ used in Section 5.2: since the coupling and measurement contributions are zero-mean, they average out, and $\hat{g}_i$ converges to the expected gradient $2\,n_{re,i}\,c_{r,i}\,\Omega_i\,\bar{\delta}_i$ at the current miscalibration.

[17] Here parameters sharing multiple rewards count extra, since their contributions add up before being squared.

C Adaptive-k estimator: derivations and validation

This appendix derives the two runtime ingredients of the adaptive-k estimator of Section 5.2 (the de-biasing of the gradient-signal estimate and the effective gain κ_{eff}) and validates the gradient empirically.

C.1 De-biasing the gradient signal

The estimators of Section 5.2 use the de-biased signal $\hat{g}^2_{i,\text{db}}$ in place of $\hat{g}^2_i$, since $\hat{g}^2_i$ is a biased estimator of $(\mathbb{E}[\nabla_{\mu_i} J])^2$. We derive that correction here:

$$\mathbb{E}[\hat{g}^2_i] = (\mathbb{E}[\nabla_{\mu_i} J])^2 + \text{Var}(\hat{g}_i) \tag{27}$$

This bias is mostly present under slow drift, when $(\mathbb{E}[\nabla_{\mu_i} J])^2$ is small relative to $\text{Var}(\hat{g}_i)$. To simplify the calculation, let $V_i \equiv \text{Var}(\nabla_{\mu_i} J)$, the per-epoch gradient variance. Adam's bias-corrected first and second moments are unbiased estimators of the actual moments (under locally stationary gradients): $\mathbb{E}[\hat{g}_i] = \mathbb{E}[\nabla_{\mu_i} J]$ and $\mathbb{E}[\hat{v}_i] = \mathbb{E}[(\nabla_{\mu_i} J)^2] = (\mathbb{E}[\nabla_{\mu_i} J])^2 + V_i$.

$$\mathbb{E}[\hat{v}_i - \hat{g}^2_i] = \big((\mathbb{E}[\nabla_{\mu_i} J])^2 + V_i\big) - \big((\mathbb{E}[\nabla_{\mu_i} J])^2 + \text{Var}(\hat{g}_i)\big) = V_i - \text{Var}(\hat{g}_i) \tag{28}$$

Now we only have to find the relation between V_i and $\text{Var}(\hat{g}_i)$; they are not equal, since $\hat{g}_i$ is a running average of gradients weighted by β_1. The relation $\hat{g}_i \propto (1-\beta_1)\sum_{j\geq 0} \beta_1^j \, (\nabla_{\mu_i} J)_{t-j}$ gives variance:

$$\text{Var}(\hat{g}_i) = (1-\beta_1)^2 \sum_{j\geq 0} \beta_1^{2j} \, V_i = \frac{(1-\beta_1)^2}{1-\beta_1^2} \, V_i = \frac{1-\beta_1}{1+\beta_1} \, V_i \tag{29}$$

Combining Eq. 28 and Eq. 29 gives $\hat{v}_i - \hat{g}^2_i = V_i - \frac{1-\beta_1}{1+\beta_1} V_i = \frac{2\beta_1}{1+\beta_1} V_i$, so the bias expressed in measured quantities is:

$$\text{Var}(\hat{g}_i) = \frac{1-\beta_1}{1+\beta_1} \, V_i = \frac{1-\beta_1}{2\beta_1} \left(\hat{v}_i - \hat{g}^2_i\right) \tag{30}$$

The de-biased signal (clamped, since both a variance and a square are non-negative), which can be used to replace $\hat{g}^2_i$, is given by:

$$\hat{g}^2_{i,\text{db}} = \max\Big(\hat{g}^2_i - \tfrac{1-\beta_1}{2\beta_1} \, \max(\hat{v}_i - \hat{g}^2_i, \, 0), \; 0\Big) \tag{31}$$

C.2 Effective gain from the feedback loop

The derivative of $\mathbb{E}\left[\delta^2_i\right]$ expands as:

$$\begin{aligned} \frac{d}{dk}\mathbb{E}\left[\delta^2_i\right] &\approx \frac{d}{dk}\sigma^2/k + \frac{d}{dk}\kappa\,(kA+B) + \frac{d}{dk}\text{lag}^2 \\ &= -\sigma^2/k^2 + \frac{d}{dk}\kappa\,(kA+B) + \frac{d}{dk}\text{lag}^2 \end{aligned} \tag{32}$$

As mentioned in Section 5.1, κ is not constant due to the influence of the Adam optimizer. Although a fixed value of κ could be used, a better approximation is to base its value on the effective per-parameter learning rate, which is tracked by Adam.

To find κ and the lag term, we treat the controller as a simple feedback loop around the optimal policy. Near the optimum, the expected gradient is proportional to the miscalibration: from Appendix B, $\mathbb{E}[\nabla_{\mu_i} J] = S_i \, \bar{\delta}_i$ with $S = 2\, n_{re}\, c_r\, \Omega$ and $\bar{\delta}_i = \mu_i - p_i^{\text{opt}}$. So on average each update pulls μ_i back towards the optimum, with a strength proportional to how far off it is. Now, let the drift be $u = p_{t+1}^{\text{opt}} - p_t^{\text{opt}}$; each Adam step then pulls $\bar{\delta}_i$ back a fraction $\lambda = \eta_{\text{eff}} S$, with η_{eff} the effective step size:

$$\begin{aligned}\bar{\delta}_{t+1} &= \mu_{t+1} - p^{\text{opt}}_{t+1} \\ &= \mu_t - \eta_{\text{eff}}(S\bar{\delta}_t + \text{noise}) - p^{\text{opt}}_t - u \\ &= \bar{\delta}_t - \eta_{\text{eff}} S\bar{\delta}_t - u - \eta_{\text{eff}} \cdot \text{noise} \\ &= (1-\lambda)\bar{\delta}_t - u - \eta_{\text{eff}} \cdot \text{noise}\end{aligned} \tag{33}$$

Since $\mathrm{Var}(u) = 0$ (drift is deterministic) and $\mathbb{E}[\text{noise}] = 0$, we now find that in the steady state $\mathbb{E}[\bar{\delta}_t] = \mathbb{E}[\bar{\delta}_{t+1}]$ and $\mathrm{Var}(\bar{\delta}_t) = \mathrm{Var}(\bar{\delta}_{t+1})$:

$$\mathbb{E}[\bar{\delta}_{t+1}] = (1-\lambda)\mathbb{E}[\bar{\delta}_t] - u \implies |\mathbb{E}[\bar{\delta}]| = \frac{u}{\lambda} = \underbrace{\frac{u}{\eta_{\text{eff}} S}}_{\text{lag}} \tag{34}$$

$$\begin{aligned}\mathrm{Var}(\bar{\delta}_{t+1}) &= (1-\lambda)^2 \mathrm{Var}(\bar{\delta}_t) + \eta^2_{\text{eff}} \mathrm{Var}(\text{noise}) \\ \mathrm{Var}(\bar{\delta})(2\lambda - \lambda^2) &= \eta^2_{\text{eff}} \mathrm{Var}(\nabla J) \quad \text{with} \quad \lambda^2 \ll 2\lambda \\ \mathrm{Var}(\bar{\delta}) &\approx \frac{\eta^2_{\text{eff}} \mathrm{Var}(\nabla J)}{2\lambda} = \underbrace{\frac{\eta_{\text{eff}}(kA+B)}{2SM_{1/2}}}_{\text{grad noise}}\end{aligned} \tag{35}$$

Now Adam gives $\eta_{\text{eff}} = \text{lr}/\sqrt{\hat{v}}$ ($\hat{v} \approx (kA+B)/M_{1/2}$). Substituting this into both gives:

$$\text{lag} = \frac{u}{\eta_{\text{eff}} S} \approx \frac{u\sqrt{kA+B}}{\text{lr}\, S\sqrt{M_{1/2}}} \qquad \text{grad noise} = \frac{\eta_{\text{eff}}(kA+B)}{2SM_{1/2}} \approx \frac{\text{lr}\sqrt{kA+B}}{2S\sqrt{M_{1/2}}} \tag{36}$$

With both terms in explicit form, the derivatives follow directly. The grad-noise scales as $\sqrt{kA+B}$ and the lag^2 as $(kA+B)$, so:

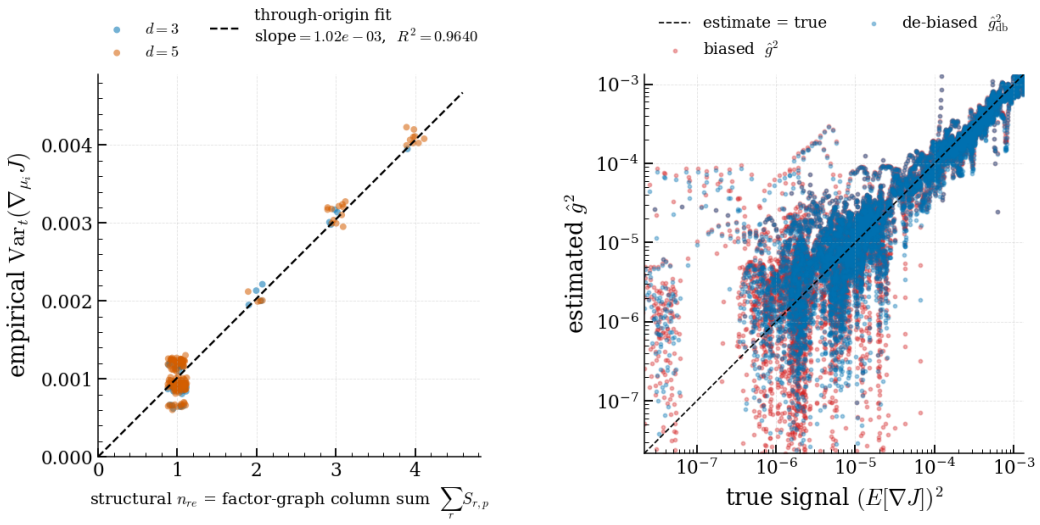
$$\begin{aligned}\frac{d}{dk}\,\text{grad noise} &= \frac{d}{dk}\frac{\text{lr}\sqrt{kA+B}}{2S\sqrt{M_{1/2}}} = \frac{\text{lr}\, A}{4S\sqrt{M_{1/2}}\sqrt{kA+B}} = \tfrac{1}{2}\kappa A, & \kappa &= \frac{\eta_{\text{eff}}}{2SM_{1/2}} \\ \frac{d}{dk}\,\text{lag}^2 &= \frac{d}{dk}\frac{u^2(kA+B)}{\text{lr}^2 S^2 M_{1/2}} = \frac{u^2 A}{\text{lr}^2 S^2 M_{1/2}} = \kappa_{\text{lag}}\, A, & \kappa_{\text{lag}} &= \frac{u^2}{\text{lr}^2 S^2 M_{1/2}}\end{aligned} \tag{37}$$

The grad-noise contributes only $\frac{1}{2}\kappa A$: because Adam's step $\eta_{\text{eff}} = \text{lr}/\sqrt{\hat{v}}$ shrinks as the noise floor $(kA+B)$ grows, this term scales as $\sqrt{kA+B}$ rather than $(kA+B)$, halving its slope. The lag has no such factor (κ_{lag} is constant in k), so it enters at full strength. Substituting both into the derivative and setting it to zero:

$$\frac{d}{dk}\mathbb{E}\left[\delta_i^2\right] = -\frac{\sigma^2}{k^2} + \left(\tfrac{1}{2}\kappa + \kappa_{\text{lag}}\right) A = 0 \quad \implies \quad k^* = \sqrt{\frac{\sigma^2}{\kappa_{\text{eff}}\, A}}, \qquad \kappa_{\text{eff}} = \tfrac{1}{2}\kappa + \kappa_{\text{lag}} \tag{38}$$

This is exactly Eq. 10 with $\kappa \to \kappa_{\text{eff}}$, so every result of Section 5.1 carries over. At runtime we can read some of these properties from the Adam state (for fixed k): the smoothed gradient is the lag signature, $\hat{g} \approx S \cdot \text{lag}$, so $u \approx \eta_{\text{eff}}\, \hat{g}$, the lr cancels, and $\kappa_{\text{lag}} = \hat{g}^2_{\text{db}}/(\hat{v}\, S^2 M_{1/2})$.

C.3 Validation



(a) Measure-term coefficient n_{re}.

(b) Gradient-signal de-bias.

Figure 7: Validation of the two estimator ingredients. **(a)** With the whole policy at the optimum (every $\bar{\delta} = 0$) the gradient variance reduces to the pure measure term, and each parameter lands on its factor-graph column sum. **(b)** At a static optimum the raw $\hat{g}^2$ floors above the true signal $(\mathbb{E}[\nabla J])^2$, while the de-biased $\hat{g}^2_{\mathrm{db}}$ follows it down toward zero. This is evaluated at only 1000 cycles per round, which inflates the measurement noise so the de-bias gap is clearly visible; under realistic cycle counts the correction is fairly small.

D Implementation details

D.1 Reinforcement-learning agent and training

The RL agent uses a simple Gaussian policy adjusted via a PGPE controller [6]. Additionally, the controller uses two techniques mentioned in Methods of Sivak et al. [1]. First, proximal policy optimization (PPO) [22] is used, which improves stability. Next, entropy regularization (ER) [23, 24] maintains exploration to allow the system to adapt to constantly changing drift. This hyperparameter can be tweaked to balance the exploration gap and the ability to track high-speed drift. Next, a replay buffer [25, 26] is used, which allows greater sample efficiency due to the agent reusing samples from previous epochs. Finally, the algorithm uses the factor graph described in Section 2, which can be used for variance reduction via gradient masking (Appendix G of Sivak, Newman & Klimov [13]).

Some additional techniques were used based on Sivak, Newman & Klimov. These techniques are not explicitly mentioned in Sivak et al., but are common and have been used in their prior work. The parameters μ and σ are both adjusted using the Adam [27] optimizer: an adaptive-learning-rate variant of gradient descent. Based on Sehnke et al. [6], symmetric sampling is also used to further reduce variance of samples; although not explicitly mentioned by Sivak et al. or Sivak, Newman & Klimov, it is a common addition to PGPE.

One aspect of the model from Sivak et al. not replicated is detector sensitivity rescaling mentioned in Supp. II.C. Here the σ values for different parameters are rescaled in order to correct for the different scales of parameters which may vary by multiple orders of magnitude. However, in the simplified error model of the simulation where all parameters are of similar magnitude, this is not needed. If this model were to be adapted for usage on physical quantum computers, detector sensitivity would still have to be implemented.

Hyperparameters used can be found in Appendix H. The policy and optimizer hyperparameters are based on Sivak, Newman & Klimov, then tuned to improve performance. The hyperparameters for the training loop are mostly taken from Sivak et al., with the exception of the replay buffer size being determined empirically.

D.2 Sparse adaptive-k PGPE

Figure 8 gives a diagram of how sparse sampling works. Algorithm 1 gives the full per-epoch procedure of the sparse, adaptive-k PGPE controller of Section 5. The reward routing, replay buffer and PGPE update follow Sivak et al.; the exploration mask and the adaptive $\hat{k}$ are this work's additions. Phase 3 is the only place the sparsity reshapes the gradient.

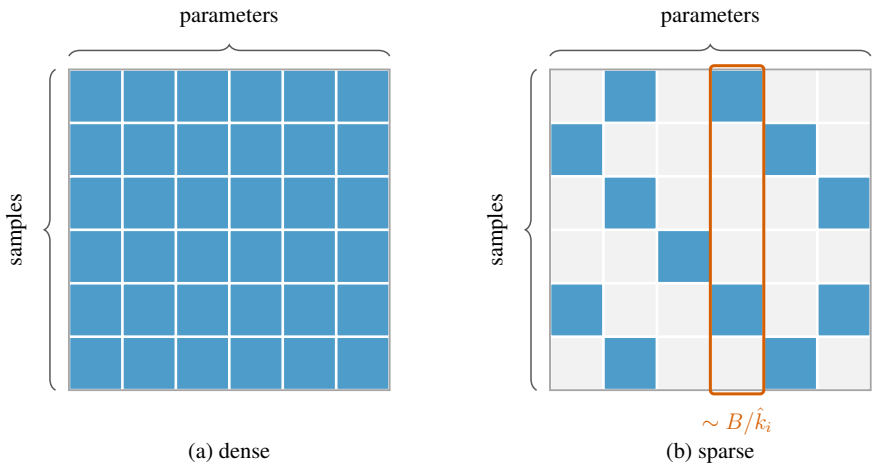


Figure 8: Sparse masking. Under dense exploration every parameter is perturbed in every sample (left); under masking with sparsity k each parameter is perturbed in only a random fraction (roughly $1/k$) of the batch's samples (right).

Algorithm 1 One epoch of sparse adaptive-k PGPE steering.

Require: policy mean μ and std σ (per parameter); Adam moments $\hat{g}, \hat{v}$; sparsity estimate $\hat{k}$; half-batch $M_{1/2}$; layout constants n_{re}, c_r, Ω, s^2

1. Build the exploration mask

for control parameter i **do**
  $m_i \leftarrow \mathrm{clip}\big(\textsc{StochasticRound}(M_{1/2}/\hat{k}_i),\ 1,\ M_{1/2}\big)$ ▷ number of pairs perturbing i
  mark i active in m_i of the $M_{1/2}$ pairs, chosen at random
end for
collect into $\mathit{mask} \in \{0,1\}^{M_{1/2}\times P}$ with $\mathit{mask}_{ji} = 1$ iff pair j perturbs i

2. Sample and evaluate

for symmetric pair $j = 1, \ldots, M_{1/2}$ **do**
  draw $\gamma_j \sim \mathcal{N}(0, \sigma^2)$ and zero inactive parameters: $\gamma_j \leftarrow \gamma_j \odot \mathit{mask}_j$
  evaluate the antithetic candidates $\mu + \gamma_j$ and $\mu - \gamma_j$; record per-detector rewards $r_j^{\pm}$
end for

3. PGPE gradient *(the only step the mask reshapes)*

route the (PPO-clipped) advantages $r_j^{\pm}$ − baseline through the factor graph to weights w_{ji}
for control parameter i **do**
  $\nabla_{\mu_i} J \ \leftarrow\ \dfrac{1}{m_i} \sum_{j:\, \mathit{mask}_{ji}=1} w_{ji}\, \dfrac{\gamma_{ji}}{\sigma_i^2}$ ▷ averaged over only the m_i pairs that perturbed i
  $\nabla_{\sigma_i} J \ \leftarrow\ \dfrac{1}{m_i} \sum_{j:\, \mathit{mask}_{ji}=1} w_{ji} \Big(\dfrac{\gamma_{ji}^2}{\sigma_i^3} - \dfrac{1}{\sigma_i} \Big) \ +\ \beta_{\text{ent}}/\sigma_i$
end for
magnitude-clip $\nabla_\mu J$, $\nabla_\sigma J$

4. Update, then re-estimate the sparsity for the next epoch

Adam-update μ, σ from $\nabla_\mu J, \nabla_\sigma J$ and refresh $\hat{g}, \hat{v}$
for control parameter i **do**
  $\eta_{\text{eff}} \leftarrow \mathrm{lr}/\sqrt{\hat{v}_i}, \quad S_i \leftarrow 2\, n_{re,i}\, c_{r,i}\, \Omega_i$
  $\hat{g}^2_{i,\text{db}} \leftarrow \max\big(\hat{g}_i^2 - \frac{1-\beta_1}{2\beta_1} \max(\hat{v}_i - \hat{g}_i^2, 0),\ 0\big)$ ▷ de-biased signal, App. C.1
  $\hat{A}_i \leftarrow 2\, \hat{g}^2_{i,\text{db}} + n_{re,i}\, s^2/(2\sigma_i^2)$
  $\kappa \leftarrow \dfrac{\eta_{\text{eff}}}{2 S_i M_{1/2}}, \quad \kappa_{\text{lag}} \leftarrow \dfrac{\hat{g}^2_{i,\text{db}}}{\hat{v}_i\, S_i^2 M_{1/2}}$
  $\hat{k}_i \leftarrow \mathrm{clip}\Big(\sqrt{\sigma_i^2/\big((\tfrac{1}{2}\kappa + \kappa_{\text{lag}})\, \hat{A}_i\big)},\ 1,\ M_{1/2}\Big)$ ▷ used in phase 1 next epoch, App. C.2
end for

E Selection pattern and error layout

Fix the number of parameters perturbed in one sample, x, each raised from the baseline rate p_ℓ to an elevated rate $p_h > p_\ell$ while the rest stay at p_ℓ. The mean error then depends only on x, not on which x parameters are chosen, so any LER difference between selections is a pure layout effect. This appendix argues that the effect of choosing parameters is small, that the lowest-LER layout is a line of perturbed gates parallel to the measured logical operator, and that uniform random selection, while not optimal, stays near-optimal in the operating regime and never becomes catastrophic.

Logical errors take place when enough errors form within a logical operator chain that the decoder corrects them incorrectly. The cheapest logical representative has weight d; the decoder fails once at least $w = \lceil (d+1)/2 \rceil$ of its d gates carry an error. Since the elevated error rate only applies to the corresponding qubit that is currently being explored, this only affects the chains that pass through the explored qubits. A chain R through $k(R)$ elevated gates has LER proportional to the following:

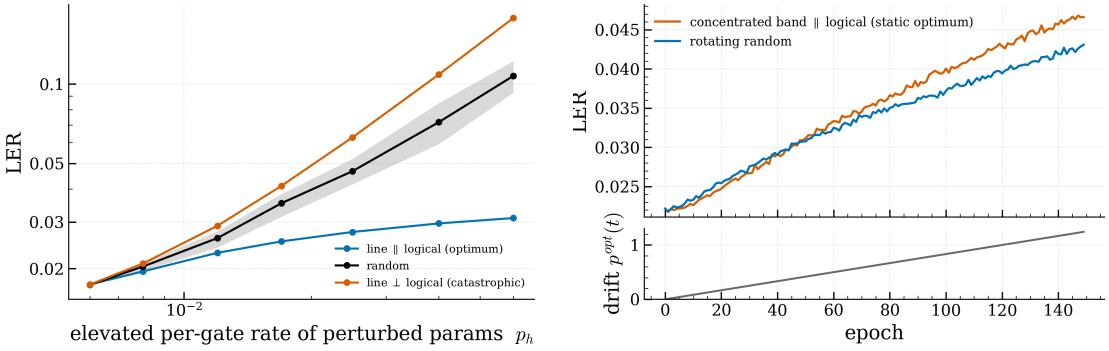
$$\mathrm{LER} \;\propto\; \sum_R p_h^{\min(k(R),\,w)}\, p_\ell^{\,w-\min(k(R),\,w)} \tag{39}$$

So each elevated gate on a chain multiplies its failure probability by $p_h/p_\ell > 1$. The layout therefore only matters through how many elevated gates the chains pass through, and the LER is lowest when the layout makes chains pass through as few elevated gates as possible (i.e. minimizing $\max_R k(R)$). Which layout achieves this depends on the measured state of the qubit, which determines what the logical operator chains look like. With $x \approx \#\mathrm{params}/d$, for the current experiment[18] the layouts are ranked from lowest to highest LER as follows (Figure 9a):

- A line parallel to the measured logical meets each of its chains at a single gate ($\max_R k(R) = 1$), so $\mathrm{LER} \sim p_h\, p_\ell^{\,w-1}$. This has the lowest LER.
- Uniform random selection scatters the gates, so a typical chain of either family meets only a few of them ($k(R) \approx wx/N$ on average) and never a whole chain. Its LER lies between the two extremes: it rises above the parallel line as the contrast grows (some of the scattered gates fall on the failure chains, whereas the parallel line's gates lie on the harmless measured logical), but it stays far below the perpendicular line, and at the operating bump it is indistinguishable from the parallel optimum.
- A line perpendicular to the measured logical is the worst: the elevated set is itself a logical-operator chain ($k(R) = d \geq w$), so that logical is effectively lost.

So the line parallel to logical operators gives the lowest LER, but there are three reasons why the random distribution is still chosen. First, although a line is optimal for one qubit state/measurement base, it is the worst case for others. The line layout is highly dependent on the state of the qubit, which would not work well for actual usage. Second, the random layout allows for easy adjustment of the sparsity compared to using a fixed pattern, which is used in Section 5. Finally, the whole effect of layout is small: at the perturbation strength we actually use, the extra error each gate adds, $\Omega\sigma^2$, is tiny next to p_ℓ, so all layouts collapse onto the same mean-determined LER and the choice of selection barely matters when compared to the loss in tracking from having all exploring parameters close together (Figure 9b). This is the empirical backing for the "LER depends only on the mean error" assumption of Section 4.1. This is why Section 5 uses uniform random selection.

[18] No gates are applied to the qubit and the qubit is measured in either the X or Z basis.



(a) Fixed mean (no steering).

(b) Under drift (steering).

Figure 9: Selection pattern and error layout ($d = 5$, $x \approx \#\text{params}/d$ gates elevated, every arrangement at the same mean error). **(a)** With no steering, the band along the line parallel to the measured logical is the lowest-LER selection (each logical chain is crossed only once), uniform random is intermediate (rising above the parallel line as the contrast grows, but staying below the catastrophic perpendicular) and the band perpendicular to it is a logical chain and is catastrophic. **(b)** Under a drifting optimum at various drift speeds (lower panel) at equal per-batch exploration budget, concentrating on the parallel line exploration (the static optimum of (a)) means the tracking worsens due to explored parameters all being close together, so it tracks the drift poorly and its LER rises above random. Even for slower drift, the gain from parallel-line exploration is only small, and does not outweigh its disadvantages.

F Estimating hardware-inspired parameter drift

In order to estimate the performance of sparse sampling under hardware-inspired drift, we need an estimate for what real drift looks like. We found no good datasets containing measurements of how this drift actually moves, so instead we will randomly generate a signal that has similar frequencies to real drift. Note that tests using this signal cannot be used to confirm performance on real quantum computers; it is merely an indication of how well the sparse sampling solution holds up under the combination of fast spikes and slow drift.

This signal is based on Figure S5a from Sivak et al. [1], which gives the power spectral density (PSD) of the logical error rate fluctuations over a run, which under a fixed policy reflects the underlying parameter drift. The frequencies in their PSD are per epoch, with matching batch size and rounds per epoch. Their PSD is approximated as proportional to C/f within the frequency band $[10^{-3}, 10^{-1}]$, or to be precise:

$$S(f) \;\propto\; \begin{cases} 1/f & f_{\mathrm{lo}} \le f \le f_{\mathrm{hi}}, \\ 0 & \text{otherwise}, \end{cases} \qquad [f_{\mathrm{lo}},\, f_{\mathrm{hi}}] \approx [10^{-3},\, 10^{-1}] \text{ per epoch} \tag{40}$$

Both edges of this range are fixed by measurements rather than arbitrarily chosen: the upper cutoff $f_{\mathrm{hi}} \approx 10^{-1}$ is the Nyquist frequency of their analysis (Sivak et al. evaluate the LER once every five epochs), so it is the highest frequency resolved, and band-limiting there discards only fast jitter that was never measured. The lower edge $f_{\mathrm{lo}} \approx 10^{-3}$ is set by the length of their runs ($\mathcal{O}(10^3)$ epochs). A $1/f$ spectrum is also the expected signature of the two-level-system defect dynamics that drive parameter drift in these devices [5], which makes this approximate signal physically plausible.

$p^{opt}(t)$ is obtained by using the inverse Fourier transform: every frequency component inside the band is given amplitude $\sqrt{S(f)}$ and an independent, uniformly random phase, and all components outside the band are set to zero. The signal is generated over a period longer than a single run, after which only a subarray of samples is used for evaluation. The overall amplitude is fixed by the constant $C = 5 \cdot 10^{-3}$ in $S(f) = C/f$, which is chosen based on Figure S5a from Sivak et al. It should be noted again that this signal cannot be used as a fully realistic drift trajectory. As an example of this, the fixed policy provides a result better than steering for this noise. This is because the value of p^{opt} stays very close to 0 and always reverts to roughly 0 after jumps (Figure 10). Despite this, this signal does provide a realistic mix of frequencies, which can be used to benchmark different steering solutions.

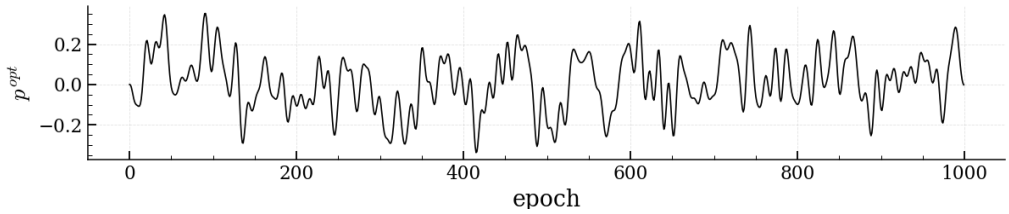


Figure 10: A synthesized hardware-inspired drift trajectory $p^{opt}(t)$ over a 1000-epoch run, generated from the band-limited $1/f$ spectrum of Eq. 40. It mixes slow wander with faster fluctuations and stays close to, and reverts toward, 0, which is why the unsteered fixed policy does deceptively well under this drift.

G Additional figures

G.1 Replication: learned-policy EDR improvement

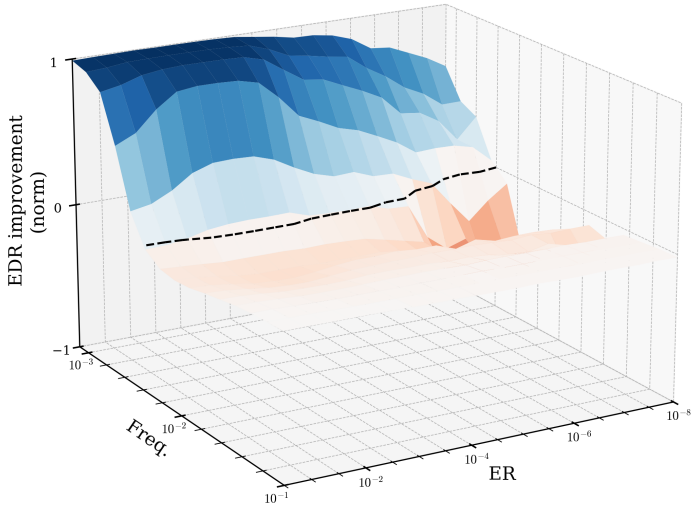
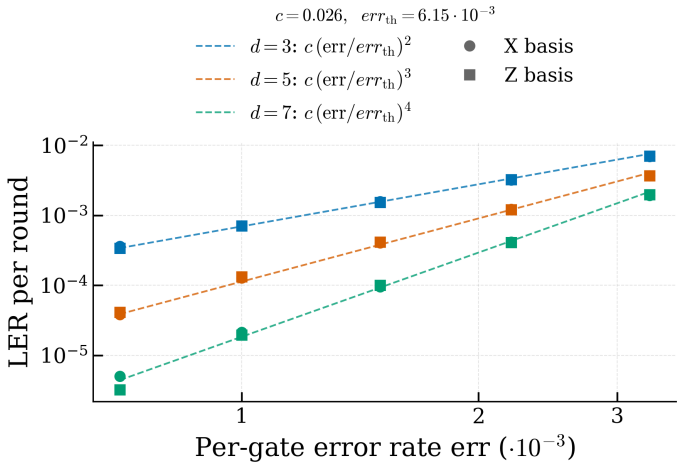
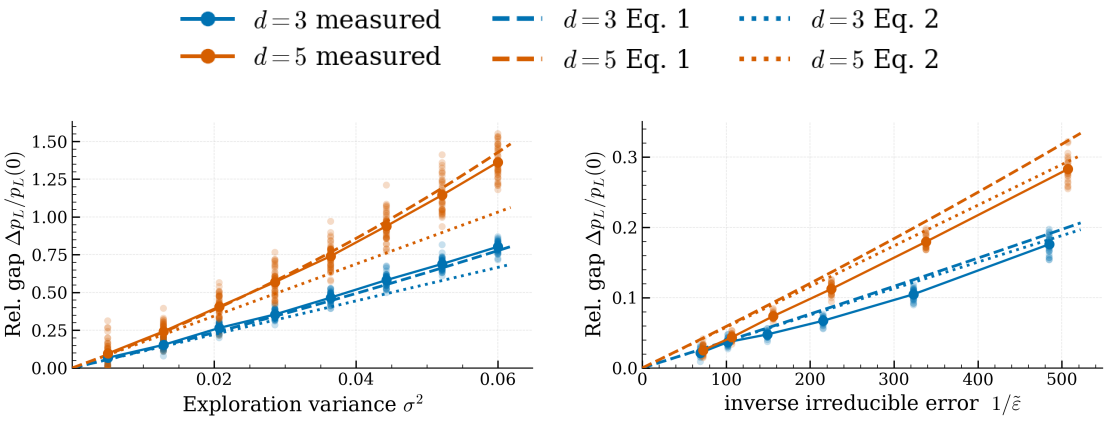


Figure 11: Replication of Fig. S8d of Sivak et al. with the independent implementation of this paper: when the learned-policy (μ) steering improves EDR over the fixed policy, for different simulation parameters. Values greater than 0 correspond to improvement. This is the learned-policy counterpart of panel (c) of Fig. 2 (stochastic policy).

G.2 Empirical confirmation of the exploration-gap derivation



(a) Power-law fit.



(b) Gap vs σ^2 ($P = 30$).

(c) Gap vs $1/\tilde{\varepsilon}$ ($P = 6$).

Figure 12: Empirical confirmation of the exploration-gap derivation. **(a)** The LER follows the power law $p_L \approx c\,(\varepsilon/\varepsilon_{th})^{(d+1)/2}$: a single shared $(c, \varepsilon_{th}) = (0.026,\ 6.15 \cdot 10^{-3})$ fits all distances and both measurement bases. **(b,c)** The relative exploration gap $\Delta p_L/p_L(0)$ against the exploration variance σ^2 (b) and the inverse irreducible error $1/\tilde{\varepsilon}$ (c), each compared to the full prediction (Eq. 1, dashed) and its linear approximation (Eq. 2, dotted); the full form matches across distances, the linear form at σ^2.

G.3 Sparsity sweep: per-cell LER traces

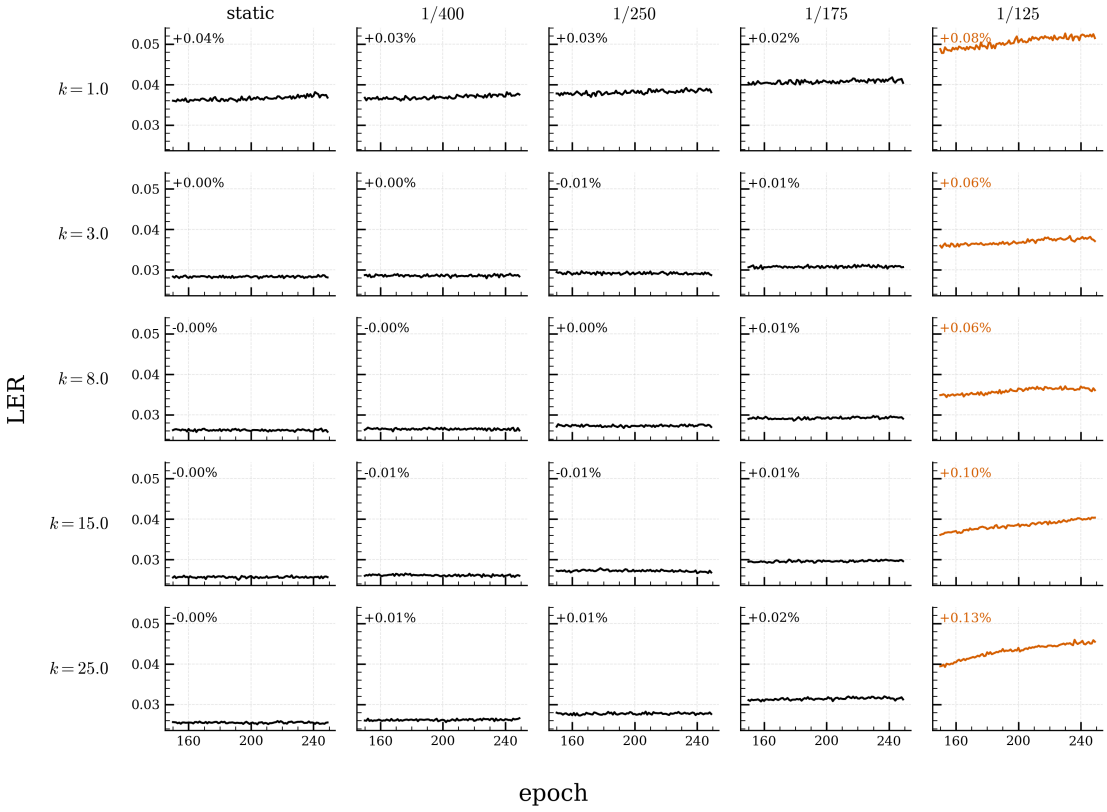


Figure 13: Last-window LER traces for a subset of the sparsity-k (rows) by drift-speed (columns) cells behind the heatmap of Fig. 3b ($d = 3$, $P = 6$). The percentage is the relative LER slope over the window; orange marks cells that fail the steady-state test (a still-rising LER, so not a converged operating point), which are the red-outlined cells excluded from the heatmap.

G.4 Step-response transient

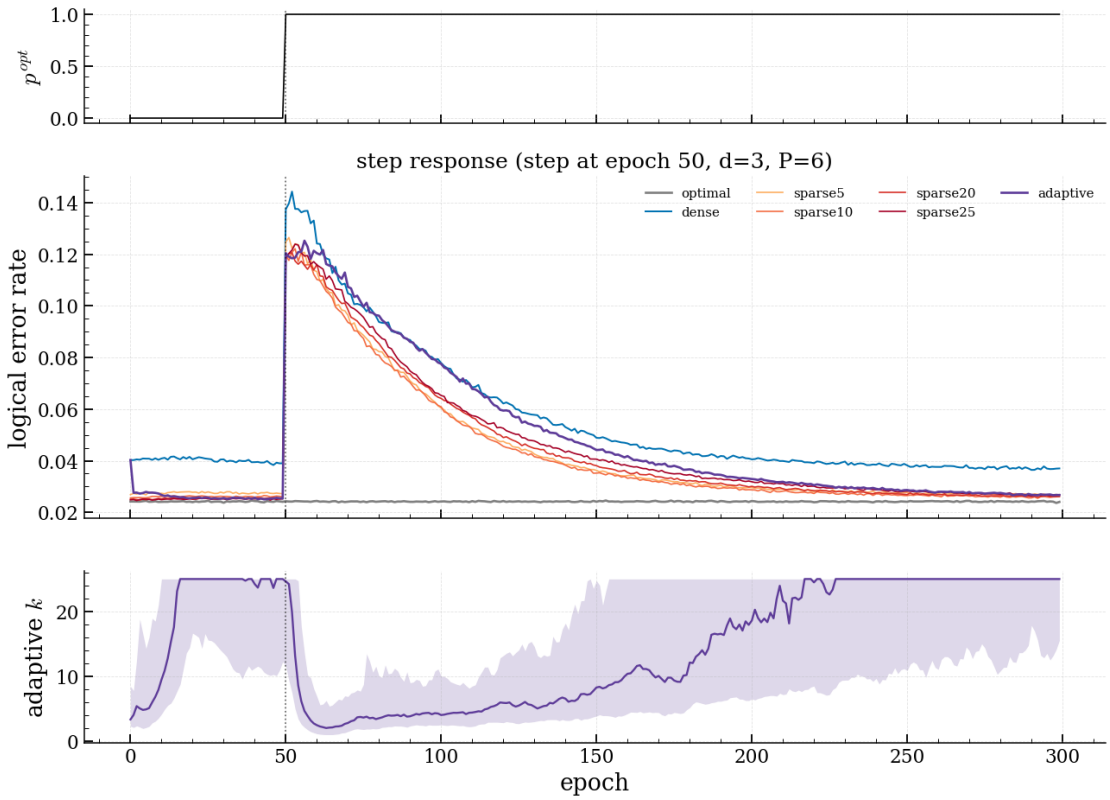


Figure 14: Per-epoch LER under a sudden step in the optimal parameters p^{opt} (a jump from 0 to 1 at epoch 50), for the optimal policy, dense exploration ($k = 1$), sparse exploration at several k, and adaptive k ($d = 3$, $P = 6$). Every policy spikes at the step and then recovers: the fixed-k sparse policies spike lower than dense and settle back near the optimal floor, while the adaptive controller drops to a low k that re-converges quickly but at a higher error rate.

G.5 Control-dimension sweep: per-cell LER and $\hat{k}$ traces

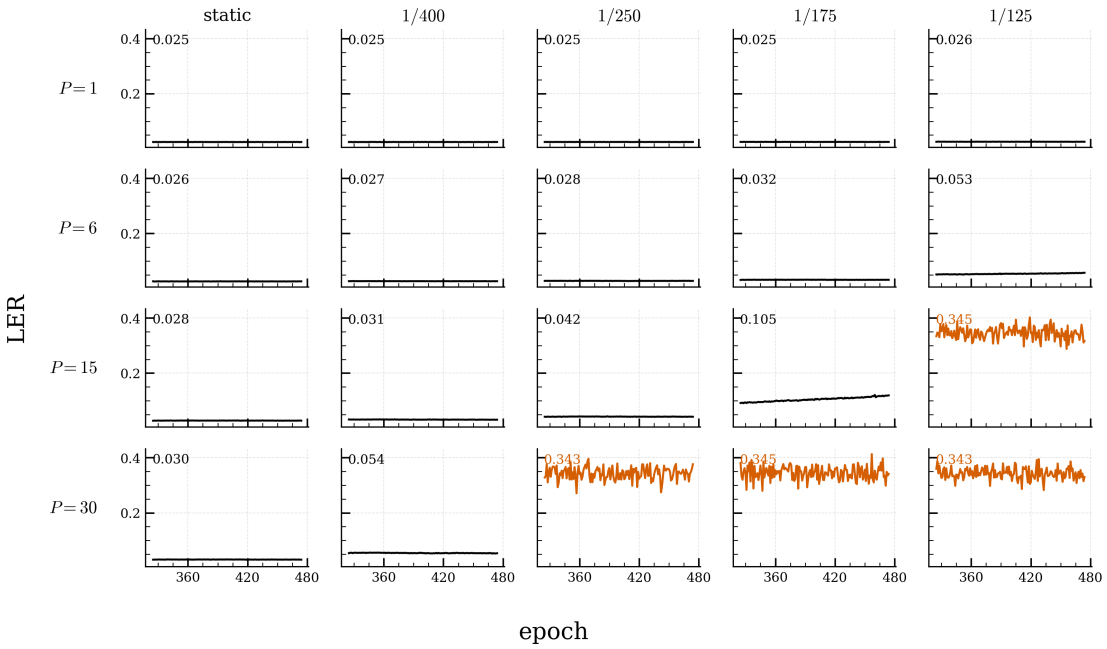


Figure 15: Per-epoch LER for a subset of the control-dimension P (rows) by drift-speed (columns) cells behind Fig. 5a ($d = 3$). Orange marks cells whose LER keeps climbing: the high-P, fast-drift corner where the loop cannot track at all.

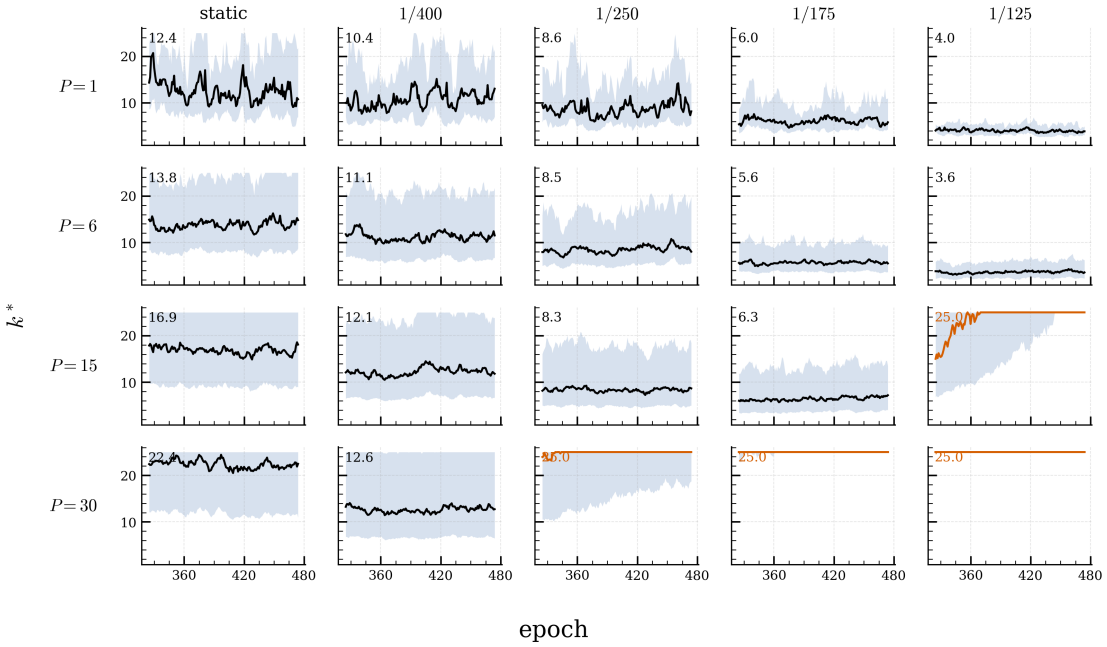


Figure 16: Chosen sparsity $\hat{k}$ (median, with IQR band) over epochs for the same cells. Orange marks $\hat{k}$ pinned at the clip $M_{1/2}$, the estimator's signature of an untrackable cell.

G.6 Gap reduction vs irreducible error

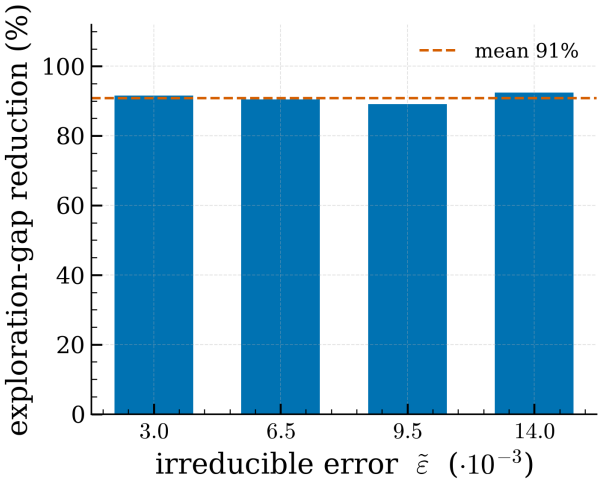


Figure 17: Exploration-gap reduction of the sparse-adaptive solution against the irreducible error rate $\tilde{\varepsilon}$ ($d = 3$, $P = 6$). The reduction stays near roughly 90% across the swept range, confirming that the benefit is essentially independent of the irreducible error rate.

H Hyperparameters

Table 2: Default hyperparameters used for PGPE training and evaluation. Values are taken from `PGPEController` and `steering`.

Parameter	**Value**
Policy	
Initial policy mean μ_0	$\mathbf{0}$ (or uniform $[-2, 2]$ for `MultiParamError`)
Initial policy std σ_0	0.45
Minimum policy std $\sigma_{\min}$	10^{-6}
Optimizer (Adam)	
Learning rate	10^{-2}
Gradient magnitude clip	0.1
PPO importance-ratio clip	0.4
Entropy regularization coefficient	10^{-3}
Value-loss coefficient	200
β_1, β_2	0.9, 0.999
Adam ε	10^{-8}
Training loop	
Batch size M	50 (paired symmetric, 25×2)
QEC cycles per candidate	$3.6 \cdot 10^4$
Replay buffer size	5 epochs