Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Convolutional neural network versus persistence for forecasting syndrome rate threshold crossings

Ashton Reddy

areddy1@exeter.edu

Phillips Exeter Academy

Article

Keywords:

Posted Date: October 30th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11555403/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Abstract

Calibration drift makes the reliability of quantum processors change over time, motivating tools that forecast degradation from data already produced during error correction. We test whether a convolutional neural network (CNN) can predict, from eight cycles of syndrome statistics, whether the next cycle's nonzero-syndrome rate in a simulated three-qubit bit-flip code will exceed a fixed threshold. Fifteen independent drifting-noise trajectories (6,000 cycles) were split by trajectory into training, validation, and test sets, so no cycle was shared between sets. On three held-out trajectories the CNN reached 89.6% accuracy and 98.0% recall for threshold crossings, while a last-cycle persistence baseline reached 91.2% accuracy. Short-horizon crossings are therefore predictable under slow drift, but the CNN shows no accuracy advantage over persistence. The results show why trajectory-disjoint evaluation and simple temporal baselines are needed when assessing machine-learning monitors for quantum error correction.

Introduction

Present-day quantum processors are noisy intermediate-scale devices whose gate, measurement, and coherence errors limit the computations they can run reliably[1]. Quantum error correction (QEC) addresses these errors by encoding logical information across several physical qubits and repeatedly measuring parity checks, called syndromes, that reveal errors without measuring the encoded data directly. Error rates are not fixed, however: device parameters drift between and during calibrations, and methods for detecting and tracking such drift are an active area of work[2].

Machine learning has been applied to QEC mainly as a decoder, inferring from syndromes which error occurred so that it can be corrected. Neural decoders have been demonstrated for the toric code[3] and small surface codes[4], and a transformer-based decoder recently outperformed leading conventional decoders on data from a superconducting processor[5]. A different question is whether the recent history of syndrome statistics can forecast an upcoming rise in the error rate, which could help schedule recalibration or flag unreliable stretches of computation before they corrupt a result.

Here we study that forecasting task in a deliberately small setting: a simulated three-qubit bit-flip code under slowly drifting noise. A two-dimensional convolutional neural network (CNN) receives eight consecutive cycles of syndrome probabilities and predicts whether the next cycle's nonzero-syndrome rate will exceed a fixed threshold. Convolutional networks suit data with local temporal structure[6]. Because neighbouring sliding windows share most of their cycles, we split data by whole noise trajectory rather than by window, and we compare the CNN with a last-cycle persistence baseline. The CNN predicts threshold crossings well above the majority-class rate, but it does not exceed persistence in accuracy. The target is a syndrome-rate proxy, not decoded logical failure.

Results

Trajectory-disjoint dataset

The dataset contains fifteen independent noise trajectories of 400 cycles each (6,000 cycles). Whole trajectories were assigned to training (nine), validation (three), and test (three) sets before eight-cycle windows were built, giving 3,528, 1,176, and 1,176 windows. Automated checks confirmed that no trajectory, feature cycle, or target cycle appears in more than one set. Class prevalence differs across sets (Table 1), so the majority baseline uses the training majority (class 1).

Table 1
Trajectory-level allocation and class distribution. Class 1 denotes a next-cycle nonzero-syndrome rate above 0.08.

Set	Trajectory IDs	Windows	Class 0	Class 1	Class 1 (%)
Train	2, 3, 4, 5, 6, 7, 8, 10, 14	3,528	878	2,650	75.11
Validation	1, 12, 13	1,176	327	849	72.19
Test	0, 9, 11	1,176	530	646	54.93
Total	15 trajectories	5,880	1,735	4,145	70.49

CNN and baseline performance

On the 1,176 test windows the CNN achieved 89.63% accuracy and 88.71% balanced accuracy, compared with 54.93% for the training-majority baseline and 91.24% for last-cycle persistence (Table 2). The CNN's ROC AUC was 0.9826, indicating strong ranking of the two classes, but ranking does not imply an accuracy advantage at the fixed decision rule.

Table 2
Performance on the 1,176-window test set.

Method	Accuracy (%)	Balanced accuracy (%)	Macro F1
Training-majority baseline	54.93	50.00	0.3546
Last-cycle persistence	91.24	91.18	0.9116
CNN (final epoch)	89.63	88.71	0.8928

The two predictors make different errors. The CNN identified 633 of 646 threshold crossings (recall 0.980) but raised 109 false alarms among 530 non-crossing cycles (Table 3). Persistence made 50 false alarms and missed 53 crossings, giving 19 more correct predictions overall.

Table 3
CNN confusion matrix and class-level metrics on the test set.

Actual class	Predicted 0	Predicted 1	Precision	Recall	F1
Below threshold (0)	421	109	0.9700	0.7943	0.8734
Above threshold (1)	13	633	0.8531	0.9799	0.9121

Variation across held-out trajectories

Performance varied with each trajectory's class balance and noise history (Table 4). Persistence matched or exceeded CNN accuracy on all three trajectories. The CNN's balanced accuracy fell to 68.31% on trajectory 9, where 87.76% of windows were threshold crossings.

Table 4
Results by held-out trajectory (392 windows each).

Trajectory ID	Class 1 (%)	CNN accuracy (%)	CNN balanced accuracy (%)	Persistence accuracy (%)
0	9.18	89.03	88.97	89.54
9	87.76	91.58	68.31	93.62
11	67.86	88.27	83.00	90.56

Discussion

Next-cycle threshold crossings were predictable from syndrome history under the simulated drift model, but a CNN did not outperform the simplest temporal predictor. This outcome follows from the setting: the noise drifts slowly relative to the cycle period, and the target is a deterministic function of syndrome probabilities already present in the input, so the most recent cycle carries most of the available signal. The CNN's higher recall for crossings, at the cost of more false alarms, may matter where a missed degradation is costlier than an unnecessary recalibration, but any such operating point would need to be chosen on validation data and tested on new trajectories.

The study also illustrates an evaluation pitfall. Sliding windows that advance by one cycle share seven of eight cycles with their neighbours, so a random window-level split places near-duplicates in both training and test sets and can inflate accuracy. Splitting by trajectory removes this leakage, and reporting a persistence baseline shows how much of the apparent skill comes from temporal autocorrelation alone. We suggest both as defaults for machine-learning monitors of QEC data.

Several limitations constrain the conclusions. Only three trajectories form independent test units, and neighbouring test windows are correlated, so no window-level confidence intervals or significance tests are reported; grouped cross-validation over all fifteen trajectories would give a firmer estimate. The

target is the observed nonzero-syndrome rate rather than decoded logical failure, each cycle re-prepares the encoded state without recovery, and the noise model omits thermal relaxation, crosstalk, and coherent drift. Future work should test regimes where the latest cycle is insufficient, such as abrupt changes or faster drift; compare sequence models such as recurrent networks and transformers against persistence on identical trajectory splits; and move to stateful QEC experiments with decoder-defined logical failure and to hardware data.

Methods

Syndrome circuit

The bit-flip syndrome circuit uses three data qubits and two ancilla qubits to measure the parity checks Z_0Z_1 and Z_1Z_2. Each cycle freshly prepares the entangled data state $(|000\rangle + |111\rangle)/\sqrt{2}$ and samples the circuit 4,000 times; no state is carried between cycles and no recovery or decoder is applied. A 00 syndrome indicates no detected parity violation. The per-cycle nonzero-syndrome rate is

$$error_rate = 1 - P(00)$$

1

Temporal dependence in the rate series arises only from the drifting noise parameters.

Noise model and drift

Simulations used Python 3.12, Qiskit 2.5.2, and Qiskit Aer 0.17.2 with density-matrix simulation[7]. The noise model applies depolarizing errors after one- and two-qubit gates and symmetric bit-flip readout errors. The one-qubit error probability p_1 is fixed at 0.001. At cycle t, the two-qubit probability p_2 is 0.01 + 0.00002t plus a Gaussian random walk with increment standard deviation 0.0002, and the readout probability p_m is 0.02 + 0.00001t plus a separate random walk with increment standard deviation 0.0001. Probabilities are clipped to [0, 1], and both random walks restart for each trajectory. For trajectory j, the noise seed is 7 + 401j and the shot-sampling seed at cycle t is 8 + 401j + t.

Dataset construction and splitting

Features are the four syndrome probabilities per cycle, ordered 00, 01, 10, 11. An example uses cycles t − 8 to t − 1 as an 8 × 4 × 1 input and has label 1 if the rate at cycle t is strictly greater than 0.08, fixed before evaluation. Complete trajectories were split before windowing: a GroupShuffleSplit (seed 42) reserved 20% of trajectories for testing, and a second GroupShuffleSplit (seed 42) assigned 25% of the remainder to validation. Cycles were sorted and checked for missing or duplicate indices within each trajectory, and each trajectory contributes 392 windows, none spanning a trajectory boundary. The allocation was fixed in advance and not selected by test performance.

Model and training

The CNN comprises a 32-filter convolution with a (3, 2) kernel and ReLU activation, (2, 1) max pooling, a 64-filter (3, 2) convolution with ReLU, a second (2, 1) max pooling, flattening, a 64-unit ReLU dense layer, dropout at rate 0.3, and two output logits. Both convolutions use “same” padding, and the model has 45,538 trainable parameters. Training used Adam (learning rate 0.001), sparse categorical cross-entropy on logits, ten epochs, batch size 32, and seed 42 for Python, NumPy, TensorFlow, and data shuffling, with deterministic TensorFlow operations. Inputs were unscaled, with no class weighting or resampling. The final epoch was used without early stopping, checkpoint selection, architecture search, or seed search; validation trajectories were monitored only, and the test set was used once after training. Software versions were TensorFlow CPU 2.20.0, Keras 3.15.1, NumPy 2.3.5, and scikit-learn 1.8.0.

Baselines and metrics

The majority baseline always predicts the training-majority class (class 1). The persistence baseline predicts class 1 exactly when the most recent input cycle's rate exceeds 0.08 and requires no fitting. We report accuracy, balanced accuracy (the mean of the two class recalls), macro F1, the confusion matrix, class-level precision, recall, and F1, and ROC AUC for the CNN.

Use of AI tools

A large language model (Claude, Anthropic) was used to help edit and restructure the manuscript text. The study design, simulations, code, and analyses are the author's own, and the author reviewed all text and takes full responsibility for the content.

Data availability

The 6,000-cycle dataset, split arrays, trained model, per-window test predictions, and metrics are available on Zenodo at https://doi.org/10.5281/zenodo.23224793 (ref. 8).

Code availability

Simulation, splitting, training, and evaluation code is available at https://github.com/ashton-reddy/quantum_computer_simulation (release v1.0, commit 22f32e9) and is archived with the data on Zenodo[8].

Declarations

Acknowledgements

The author thanks Dr. Ananda Gunawardena of the Department of Computer Science, Princeton University, for mentorship and guidance on this project.

Author contributions

A.R. conceived the study, wrote the simulation and analysis code, performed the experiments, analysed the results, and wrote the manuscript.

Competing interests

The author declares no competing interests.

References

1. Preskill, J. Quantum computing in the NISQ era and beyond. *Quantum* **2**, 79 (2018).
2. Proctor, T. et al. Detecting and tracking drift in quantum information processors. *Nat. Commun.* **11**, 5396 (2020).
3. Torlai, G. & Melko, R. G. Neural decoder for topological codes. *Phys. Rev. Lett.* **119**, 030501 (2017).
4. Varsamopoulos, S., Criger, B. & Bertels, K. Decoding small surface codes with feedforward neural networks. *Quantum Sci. Technol.* **3**, 015004 (2017).
5. Bausch, J. et al. Learning high-accuracy error decoding for quantum processors. *Nature* **635**, 834–840 (2024).
6. Kiranyaz, S. et al. 1D convolutional neural networks and applications: a survey. *Mech. Syst. Signal. Process.* **151**, 107398 (2021).
7. Qiskit contributors. Building noise models. Qiskit Aer documentation https: (2026). //qiskit.github.io/qiskit-aer/tutorials/3_building_noise_models.html
8. Reddy, A. quantum_computer_simulation: v1.0 paper reproducibility archive. Zenodo https://doi.org/10.5281/zenodo.23224793 (2026).