Quest Journals
Journal of Software Engineering and Simulation
Volume 12 ~ Issue 10 (October 2026) pp: 07-11
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Machine-Learning-Based Decoding for Quantum Error Correction: A Simulation-Based Comparison Using a Five-Qubit Repetition Code

Darsh Daivik Madhu

ABSTRACT
Noise can change the state of a qubit and cause errors in a calculation. Quantum error correction (QEC) protects information by storing it across several physical qubits and using check results, called a syndrome, to help identify likely errors. This project compares three ways to decode a five-qubit repetition-code simulation: majority voting, a probability-based maximum a posteriori (MAP) lookup, and a neural network. The methods were tested with independent bit flips and with occasional three-bit bursts. All three gave the same error rate under independent noise. Under burst noise, the neural network matched the MAP decoder and had far fewer errors than majority voting. The neural network took longer to run, and the network trained on independent noise did not work well when tested on burst noise without retraining. The results show that a neural network can help under some conditions, but it is not automatically better than a conventional method that already uses the correct noise model.

*Received 26 Oct., 2026; **R**evised 30 Oct., 2026; **A**ccepted 06 Nov., 2026 © The author(s) 2026.*
P**ublished with open access at **www.questjournals.org

I. Introduction

Errors can occur when qubits interact with their surroundings or when operations are imperfect. Quantum error correction protects information by spreading it across several physical qubits. It uses parity checks to produce a set of results called a syndrome. A decoder uses these results to choose the most likely correction.

Conventional decoders follow mathematical rules or use a known noise model. A machine-learning decoder learns from examples of errors and their syndromes. This project tests whether a small neural network can make accurate decisions under two types of noise and whether it can still work when the noise changes. The experiment is a classical simulation; no quantum processor was used.

The original contribution of this project is a controlled comparison of three decoders on the same five-qubit model. It tests both independent and burst noise and also checks whether a network trained on one type of noise can handle the other. The aim is not to propose a new decoder, but to identify when a learned decoder helps and where it has limits.

II. Background and Previous Research

The five-bit repetition code stores logical 0 as 00000 and logical 1 as 11111. A majority vote can correct up to two bit flips because the original value is represented in three or more of the five positions. This code only protects against bit flips; it does not protect against phase flips, so it is not a complete quantum error-correction code [1]. The five qubits and four neighbouring checks used in this study are shown in Figure 1.

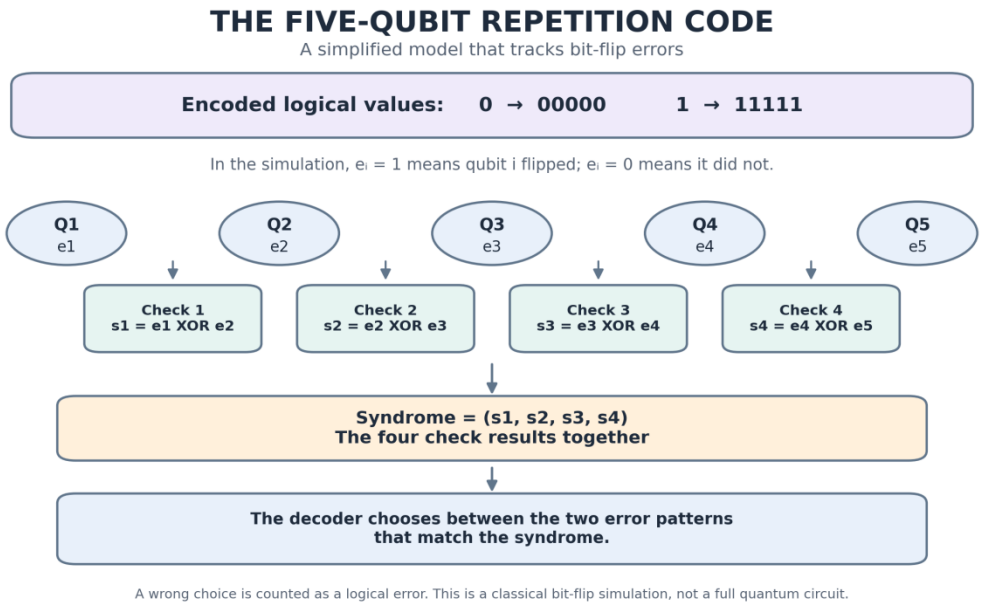


Figure 1. Five-qubit repetition code and its four neighbouring parity checks.

Decoding becomes more challenging for larger codes. Higgott's PyMatching package implements minimum-weight perfect matching (MWPM), a conventional method that finds a low-weight correction consistent with the check results. It also includes a local-matching option that limits which check events can be paired; the paper reports that this can reduce computation while keeping similar decoding performance [2]. Bausch et al. developed AlphaQubit, a neural network decoder for the surface code. It outperformed other leading decoders on experimental data from Google's Sycamore processor for code distances three and five. The authors also tested it on simulated systems up to code distance 11 [3]. Recent studies continue to examine the balance between decoding accuracy and speed, including whether neural decoders can operate in real time [4], [5]. These studies use larger and more realistic systems than the small simulation in this project.

III. Methodology

Each simulated error was stored as five binary values: 1 meant a bit had flipped and 0 meant it had not. The program compared neighbouring positions to calculate four parity checks: $s_i = e_i$ XOR e_{i+1}, for i = 1, 2, 3, 4. (XOR gives 1 when the two values differ and 0 when they are the same.) The four check results form the syndrome. Each syndrome is consistent with two possible error patterns: one pattern and the pattern with all five bits flipped. The decoder must choose between these two possibilities; a wrong choice produces a logical error.

The project compared three decoders. Majority voting chose the error pattern with fewer flipped bits. The maximum a posteriori (MAP) lookup, which is a probability-based method, calculated how likely each of the two patterns was under the selected noise model and chose the more likely one. The neural network learned from labelled examples which of the two patterns was more likely. The network was a small multilayer perceptron with four inputs (the four syndrome bits), two hidden layers containing 16 and 8 units, and one output. It used the ReLU activation setting, the Adam training method, and early stopping.

Two noise conditions were used. For independent noise, each of the five bits flipped independently with probability p = 0.05. For burst noise, background flips occurred independently with p = 0.03. In addition, with probability q = 0.15, a burst flipped three neighbouring bits. Its starting position was chosen at random, with equal probability, from the three possible positions. The program combined a burst with the background errors using XOR, so a bit that was flipped in both steps ended up unflipped. For each condition, a separate neural network was trained on 100,000 generated examples and tested on a different set of 300,000 examples. All three decoders were tested on the same test examples. The network trained on independent noise was also tested on the burst-noise set without being retrained.

The main measure was the logical-error rate: the share of test examples for which a decoder chose the wrong one of the two possible patterns. Accuracy was one minus this rate. The program also calculated 95% Wilson confidence intervals, which give an uncertainty range for each measured error rate. To compare speed, it first ran a warm-up, then timed five runs on the same batch of 50,000 test syndromes. It divided the median batch time by the number of samples to report median microseconds per sample. Network training time was recorded separately.

The simulation was written in Python. It used NumPy to create and process the error patterns, pandas to save the results, and scikit-learn to train the neural network. Fixed random seeds were used so the samples and training process could be repeated under the same software settings. During training, 10% of the examples were set aside to check the network's progress. Early stopping ended training when performance on this validation set stopped improving; the code allowed up to 500 training rounds and stopped when the validation score did not improve for 25 rounds. Figure 2 summarises the simulation workflow.

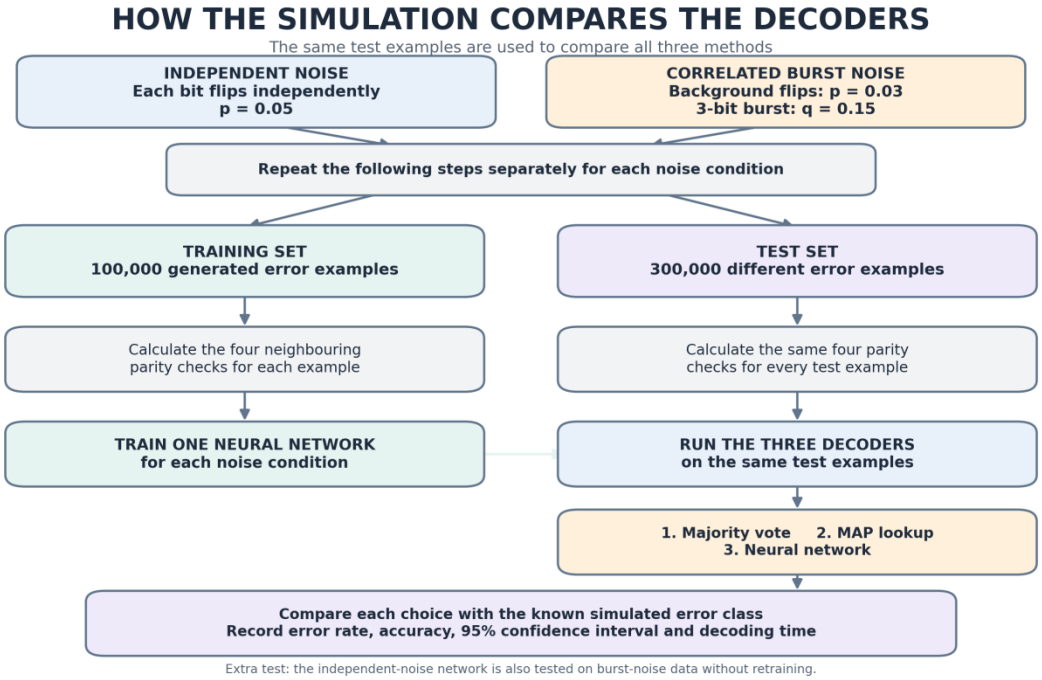


Figure 2. Workflow used to generate, train, test, and compare the three decoders.

IV. Results

Under independent noise, all three decoders made 333 errors out of 300,000 tests (0.111%; 95% CI 0.100–0.124%), for an accuracy of 99.889%. Under burst noise, majority voting made 41,163 errors (13.721%; 95% CI 13.598–13.845%). The MAP decoder and the network trained on burst noise each made 4,019 errors (1.340%; 95% CI 1.299–1.381%). This is 90.2% fewer errors than majority voting. When the independent-noise network was tested on burst noise without retraining, it made 41,163 errors (13.721%), the same count as majority voting. Figure 3 shows these results.

In the run used to produce the accompanying results file, median decoding time per sample was 0.0173 microseconds for majority voting under independent noise and 0.0171 under burst noise; 0.0431 and 0.0441 for the MAP lookup; and 0.0980 and 0.0877 for the neural network trained on each matching condition. The independent-noise network took 0.0880 microseconds per sample when tested on burst noise. Network training took 3.21 seconds for the independent-noise network and 3.40 seconds for the burst-noise network. These measurements depend on the computer and software and are not quantum-hardware decoder latencies.

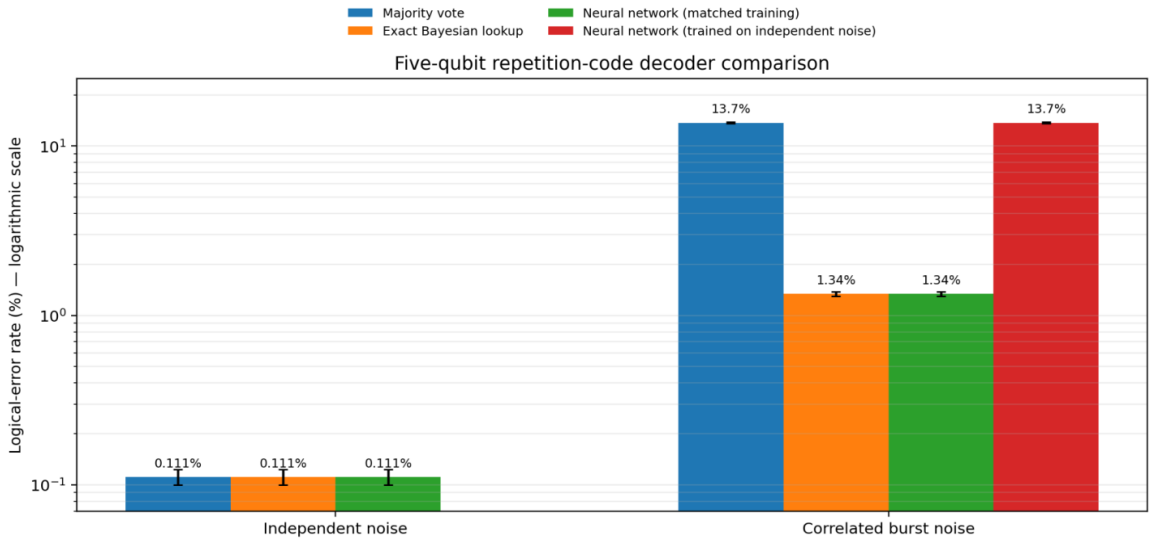


Figure 3. Comparison of logical-error rates for the five-qubit repetition-code decoders under independent and correlated burst noise. The red bar represents the neural network trained on independent noise and tested on correlated noise, illustrating the effect of distribution shift.

V. Discussion and Limitations

Under independent noise, majority voting already chose the most likely of the two possible patterns, so the neural network did not improve accuracy. Burst noise changed which pattern was more likely for some syndromes. A pattern with more flipped bits could sometimes be more likely because several flips arrived together. A network trained on burst-noise examples could learn these cases, and its error rate matched the MAP lookup. It did not beat MAP, however, because MAP was given the exact probabilities used to create the simulated noise. With those probabilities known, the lookup could make the same best choices without learning from examples.

The burst-trained network learned a relationship between the syndrome and the likely error pattern from its training examples. The independent-noise network had learned from a different noise pattern. When it was tested on burst-noise examples, it had not been trained to expect groups of neighbouring flips. It did not retrain during testing, so its error rate rose to 13.721%. This is an example of distribution shift: the test examples no longer followed the noise pattern used to train the network.

A separate five-seed check used 25,000 training examples and a shared 100,000-example test set. Burst-trained networks had error rates from 1.409% to 1.442%; four matched the MAP baseline on that shared test set, while one was slightly worse. Because this check used fewer examples than the main experiment, it is supporting evidence rather than the main result.

This was a classical simulation of bit flips, not a full quantum-circuit or hardware experiment. It did not model phase flips, faulty parity measurements, leakage, or repeated syndrome measurements. The burst-noise pattern was created by the program and may not represent every type of correlated noise in real hardware. The generalisation check was one-way: the independent-noise network was tested on burst noise, but the reverse test was not part of the main experiment. Timing results depend on the Python implementation and computer used. The MAP decoder also had an advantage: it was given the exact probabilities used by the simulation, which assumes that the noise model is already known.

VI. Conclusion

The neural network did not improve accuracy under independent noise. Under burst noise, it made far fewer errors than majority voting and matched the MAP lookup. It was slower, and the network trained on independent noise did not handle burst noise well. These results do not show that machine learning is always better. They show that its value depends on the noise pattern, the examples used for training, and the conventional decoder used for comparison. Future work could test larger codes, more realistic noise, and repeated syndrome measurements.

References

[1] IBM Quantum Learning, "Error correction II: When is error correction the right tool for quantum advantage?" IBM Quantum Learning. [Online]. Available: https://quantum.cloud.ibm.com/learning/en/courses/tools-for-quantum-advantage/error-correction-ii. [Accessed: Oct. 30, 2026].

[2] O. Higgott, "PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, Art. no. 16, 2022. doi: 10.1145/3505637.

[3] J. Bausch et al., "Learning high-accuracy error decoding for quantum processors," *Nature*, vol. 635, pp. 834–840, 2024. doi: 10.1038/s41586-024-08148-8.

[4] G. Yan, S. Li, and Y. Du, "Rethink the role of neural decoders in quantum error correction," in *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, vol. 306, pp. 143726–143759, 2026. Available: https://proceedings.mlr.press/v306/yan26g.html.

[5] C. Lee, T. Hur, and D. K. Park, "Scalable neural decoders for practical real-time quantum error correction," *npj Quantum Information*, published online Oct. 28, 2026. doi: 10.1038/s41534-026-01403-y.