Rotated Surface-Code Memory Simulation with Qiskit Front End, Stim Backend, and MWPM Decoding

Abstract

This work provides a reproducible simulation stack for surface code memory experiments. Circuits are constructed in Qiskit, simulated in Aer with circuit-level noise or exported to Stim for a more efficient sampling approach, then decoded with a local heuristic or MWPM (minimum-weight perfect matching) decoding through PyMatching. A command line interface (CLI) allows for sweeps through code distance, noise levels, and decoders with logged results based on deterministic seeds, git SHA, wall time, and bootstrap confidence intervals. Preset experiments also enable figure generation across the depolarizing and biased noise regimes. We discuss the design choices made, the observed trends, noise models and decoding strategies, and imperfections from the process while also giving a roadmap to better-fitting decoders and scalable experiments.

1. Motivation and Goals

Surface codes are a leading architecture for fault-tolerant quantum computing as they use local stabilizer measurements relative to large code distances so logical errors are suppressed. To prepare for algorithmic needs, hardware accommodations, and decoder expansions, a basic yet extensible pipeline that determines how physical noise levels correlate to logical error rates for various distances and decoders is a necessity. This work aims to:

- Establish a compact, extensible, and easily tested simulation stack for rotated surface code memory experiments.

- Allow paths for teaching versus high-throughput (Qiskit/Aer vs. Stim) sampling.

- Expose different decoders with MWPM as the baseline through PyMatching.

- Ensure any figure and corresponding CSV can be rerun based on a logged seed, command, and git SHA.

- Present preset experiments and a concise paper for easy onboarding and reproducibility.

2. Background

Surface codes are rotated with qubits positioned on a square lattice with staggered X and Z stabilizers. A memory experiment measures stabilizers over multiple rounds and checks for logical X or Z flips over the duration as syndromes change with time. Logical failure probability

is a function of physical gate noise, measurement noise, code distance, and decoder performance. MWPM is the standard decoder for surface codes as it pairs detection events in space and time to determine which combination of errors is most likely. Stim provides fast sampling of stabilizer circuits or detector error models, while Qiskit/Aer offers a more flexible approach for advanced circuit building.

3. Architecture

The codebase under `src/surface_code_sim/` is organized as follows:

- `qiskit_frontend/`: rotated surface-code circuit builders, Aer noise models, and an Aer sampler that returns detection events.
- `stim_backend/`: Stim circuit generation and sampler for faster Monte Carlo sweeps.
- `decoders/`: local heuristic decoder and PyMatching-based MWPM decoder.
- `experiments/`: preset sweeps and figure generation.
- `plotting/`: logical-vs-physical error plotting with bootstrap confidence intervals.
- `cli.py`: Typer-based sweep interface with deterministic seeding and logging.
- `profiling.py`: quick timing of Aer vs Stim samplers.
- `notebooks/qiskit_demo.ipynb`: a small end-to-end demonstration.
- `paper/`: this document.

Tests cover layout construction, noise models, samplers, decoders, plotting, and CLI integration, and the full suite passes under Python 3.11 with pinned dependencies.

4. Circuit Structure

Logical distance $d \in \{3,5,7,9\}$ means data qubits positioned in a $d \times d$ grid; for X stabilizers, even face (row+col) parity; for Z stabilizers, odd parity is maintained. The following operations occur per round:

1. Reset Z ancilla, CX from data to Z ancilla, measure Z ancilla into dedicated classical bits.
2. Reset X ancilla, apply H, CX from X ancilla to data, apply H to measure X ancilla.

Thus, the classical registers store stabilizer measurements per round so time-differentiated detection events can be computed; this aligns with a standard rotated-code memory experiment and should make gate scheduling clear for Aer and Stim backends.

5. Noise Models

Two families of gate-noise (and readout noise) apply.

- Depolarizing: symmetric depolarizing gate on one- and two-qubit gates; symmetric or asymmetric readout flip.

- Biased Pauli: independent probabilities of px, py, pz on one- and two-qubit gates (tensor-product channel), optional asymmetric readout flip.

For Aer, noise is applied via Qiskit noise models. For Stim, depolarizing uses DEPOLARIZE1/2; biased Pauli uses PAULI_CHANNEL_1 (three parameters). For now, Stim assumes readout flip is symmetric; if readout is asymmetric then it is rejected.

6. Paths of Sampling

6.1 Aer Sampler

Transpiles the circuit for AerSimulator with the noise model and runs a fixed number of shots at a fixed seed for raw memory bits, reshaped into arrays (shots, rounds, stabilizers) for X/Z measurements. Detection events occur via temporal parity (current round xor previous round; first round propagated).

6.2 Stim Sampler

Builds the comparable Stim circuit (reset/CX/measure, gate noise, readout flips), compiles a seeded sampler, then reshapes the same. Faster for larger sweeps and shot numbers.

Both paths produce deterministic results for fixed seeds for regression testing and reproducibility.

7. Decoders

7.1 Local Heuristic

Any Z (X) detection event triggers the logical X (Z) flag across rounds/stabilizers. Conservative for rapid baseline checks, not optimal.

7.2 MWPM via PyMatching

Detection events are flattened to detector integer indices. A boundary node is added so each detector connects at unit weight to the boundary; PyMatching's decode_batch returns parity of faults. Logical X/Z flags are the parity of matched faults on the other stabilizer type. This boundary-star graph was chosen for speed/ease because it does not require a geometry build step; next time, a Stim DEM for the rotated code should be ported into PyMatching for geometry-aware weights [Fowler2012; Higgott2021].

7.3 Validation and Testing

Unit tests assert layout build, sampler determinism (Aer and Stim), noise model configuration, decoder behavior, CLI CSV export, plotting, and seed normalization. All tests run with pytest under Python 3.11 and pinned dependencies for cross-device stability.

8. CLI, Logging and Confidence Intervals

The CLI (`python -m surface_code_sim.cli sweep`) allows multi-value sweeps for distance, p or px, py, pz, backend (Aer/Stim), decoders (Local/MWPM), rounds, shots, readout-specific parameters, seeds and parallel jobs. Within one run, the following logs:

- run_id, git_sha (auto-resolved if run in a git directory), seed
- distance, rounds, shots, decoder, backend
- noise (p or px, py, pz, readout errors)
- logical_error_rate, ci_low, ci_high (bootstrap, default 95%)
- wall_time_seconds, timestamp_utc

Bootstrap confidence intervals are generated per run and saved to the CSV. Figure commands are logged for reproducibility.

8.1 Data Schema and Logging Conventions

CSV columns are standardized (see `CSV_FIELDS` in code) with run metadata, noise parameters, decoder/backend identifiers, logical error rate, confidence bounds, and timing. Figure logs include the command, seed, figure path, and notes. Seeds increment over sweep loops. The git_sha is recorded automatically when in the repo. Wall time is recorded per run to track performance.

8.2 Environment and Tooling

Dependencies are pinned in requirements.txt (Qiskit packages plus Aer/Stim/PyMatching/NumPy/SciPy/pandas/matplotlib/Typer). Dev extras include pytest/ruff/pre-commit. Python 3.11.x and .pre-commit-config.yaml hooks for lint/format are included. Tests assume the package is installed editable.

8.3 Methodological Decisions Made By The Author

- Rounds/Shots: Presets use 3 rounds and 200 shots because runtime on my Mac is seconds; fine for sanity checks but not threshold estimation. Increase shots/rounds for research-grade curves.

- MWPM Graph: Boundary-star matching used for speed/ease (no geometry build). Next time: add a Stim DEM for rotated code into PyMatching to provide spatial/temporal weights and bias.

- Backends: Preferential use of Stim for larger sweeps; Aer for asymmetric readout/circuit-level explorations.

- Hardware: Development/runs on a MacBook (ARM) with Python 3.11 pinned wheels; no GPU dependence.

- Noise Parameters: Preset depolarizing sweeps $p \in \{0.001, 0.002, 0.005, 0.01\}$. Preset biased noise uses px=0.05; py=0; pz=0.15 to emphasize Z-type failures.

9. Experiments

9.1 Preset Sweeps (Stim + MWPM)

- Depolarizing: $d \in \{3,5,7,9\}$, $p \in \{0.001, 0.002, 0.005, 0.01\}$, rounds = 3, shots = 200.

- Biased: $d \in \{3,5,7,9\}$, px = 0.05, py = 0.0, pz = 0.15, rounds = 3, shots = 200.

- Backend: Stim; Decoder: MWPM.

Outputs: `experiments/presets.csv`, `figs/preset_dep.png`, `figs/preset_biased.png`, logs in `experiments/run_commands.log` and `experiments/fig_commands.log`.

9.2 Expected Trends

- Depolarizing: logical error should decrease with distance at fixed p; increase with p at fixed distance. With only 200 shots, curves are coarse and do not have well-resolved threshold crossings.

- Biased: Z-type failures will dominate due to larger pz; logical Z rates should exceed X rates given the MWPM reduction used here.

9.3 Reproducibility

Seeds are deterministic per combination; git SHA is noted; wall times are logged. Dependency pins ensure repeatable environments, and tests validate the full stack.

9.4 Execution Details

The preset sweep runs 20 (16 depolarizing; 4 biased). Wall times are short (seconds per run on a laptop) given 200 shots and three rounds per run. Outputs append to `experiments/presets.csv`; figures are written to `figs/`. Commands to make the figures and runs are appended to logs for auditability. Users can change shots and rounds from the CLI flags; confidence intervals will broaden or narrow accordingly.

9.5 Measuring Logical Outcomes

Logical outcomes are determined per shot: a logical event occurs if either inferred logical flag for X or Z is nonzero. Logical error is the mean over all shots. Confidence intervals use bootstrap resampling (default 5,000 resamples) at 95%. CIs may be wide if shot counts are small; increasing shots narrows variance.

10. Results

The preset figures (in `figs/`) show expected monotonic trends:

- For depolarizing noise, at fixed p the logical error rate drops as distance increases; as p increases, the curves steepen.

- For biased noise, Z logical rates dominate. The MWPM reduction is simplified but does show distance suppression (not optimized for bias-aware decoding).

CSV outputs report bootstrap confidence intervals. With modest shot counts, CIs are relatively wide.

Results cite their sources explicitly: see `figs/preset_dep.png`, `figs/preset_biased.png`, and the numeric rates in `experiments/presets.csv`.

11. Performance and Profiling

`profiling.py` times small Aer vs Stim runs at distances 3 and 5 with 100 shots and two rounds, using the local decoder. Stim is faster for larger sweeps; Aer is convenient for circuit-level exploration and asymmetric readout studies. Future profiling should time MWPM and scale with distance/shots for planning larger runs.

12. Related Work

Surface-code decoding and threshold estimation have been extensively reviewed. Fowler et al. [Fowler2012] assessed threshold performance for surface codes and decoders under circuit-level noise, setting benchmarks. Gottesman [Gottesman2009] discussed quantum error correction broadly. Stim [Gidney2021] provides fast stabilizer simulation and detector error models for Monte Carlo studies. PyMatching [Higgott2021] offers matching for stabilizer codes. Our work leans on these sources and aims to simplify small-scale explorations for reproducibility and expansion.

13. Conclusion

We presented a reproducible surface-code simulation using Qiskit and Stim backends with deterministic seeding, logged metadata, bootstrap confidence intervals, and an MWPM decoder via PyMatching. Preset experiments and plotting provide quick feedback on logical error dynamics across distance and noise regimes. The next worthwhile improvements are a geometry-aware MWPM graph, larger threshold studies, and extended noise models.

14. References

- [Fowler2012] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, “Surface codes: Towards practical large-scale quantum computation,” Phys. Rev. A 86, 032324 (2012). DOI: 10.1103/PhysRevA.86.032324.

- [Gottesman2009] D. Gottesman, “An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation,” arXiv:0904.2557 (2009). https://arxiv.org/abs/0904.2557.

- [Gidney2021] C. Gidney, “Stim: a fast stabilizer circuit simulator,” arXiv:2103.02202 (2021). https://arxiv.org/abs/2103.02202.

- [Higgott2021] P. Higgott, “PyMatching: a Python package for decoding quantum codes with minimum-weight perfect matching,” arXiv:2105.13082 (2021). https://arxiv.org/abs/2105.13082.