Executive Summary
Classical Simulation of Quantum Circuits with Pauli Propagation

Steven Ma (ym432)
Supervised by Prof. Hamza Fawzi

June 2025

Word Count: 998

1 Introduction

Simulating the behaviour of quantum systems is one of the most challenging problems in science, with far-reaching implications for physics and materials science. Classical computers become fundamentally limited by exponential memory requirements. Quantum algorithms running on quantum computers are theoretically capable of overcoming these barriers. However, the necessity of quantum computers is justified only if they can surpass the most advanced classical algorithms; quantum advantage is meaningful only when classical simulation has been pushed to its limits. This project presents a Python implementation of the Pauli propagation algorithm [1] (`pauli_propagation`) and its simulation results, aimed at pushing classical simulation to a new limit.

2 Method Overview

Pauli Propagation Algorithm

A central goal of quantum simulation is to predict the expectation of an observable after a quantum system evolves. This evolution can be modelled by a quantum circuit: a programmable sequence of quantum operations (gates) applied to a system of qubits. By analyzing the expectation values produced by a quantum circuit, one can extract detailed information about the evolution and physical properties of the corresponding quantum system.

The Pauli propagation algorithm provides an efficient approach to estimating these expectation values in large quantum circuits. It does so by decomposing any observable O into a sum of simple Pauli operators (I, X, Y, Z), then tracking how each operator evolves as the circuit acts in the Heisenberg picture (i.e., evolving the observable rather than the quantum state). To avoid exponential computational growth, the method discards negligible high-weight operators at every step, where the Pauli weight threshold k is a user-defined parameter. Here, Pauli weight simply counts how many qubits the operator acts on nontrivially (e.g., $Y_2 \otimes I_1 \otimes X_0$ has weight 2). Increasing k preserves more detail but also increases computational time and memory requirements. If no operators are discarded, the algorithm yields an exact simulation. For certain a class of circuits (specifically, the circuits described in Section 3.1), the mean squared error (MSE) of this expectation estimate is upper bounded by theory.

This core idea is schematically illustrated in Figure 1: as the observable is back-propagated through each circuit layer, a single operator may branch into multiple operators of varying weights, but only those below a chosen weight threshold are retained at each step. The final set of operators is combined with the initial state to estimate the expectation value.

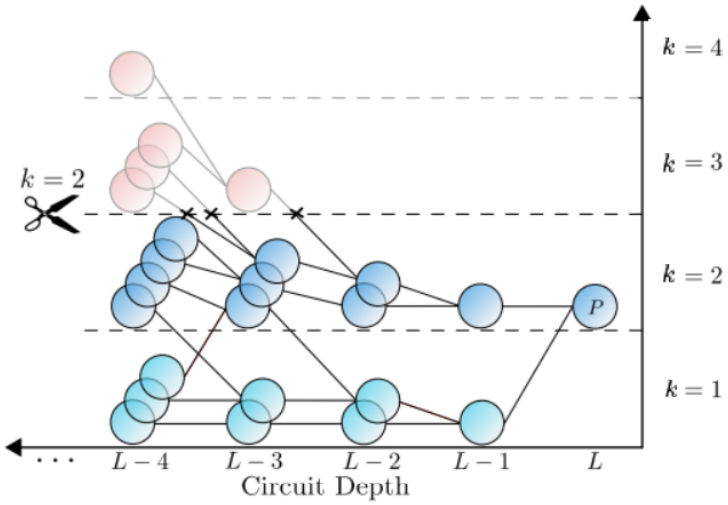


Figure 1: Schematic of Pauli propagation with weight truncation ($k = 2$). Adapted from [1].

Monte Carlo Error Estimation

While the Pauli propagation algorithm enables estimation of observable expectations in quantum circuits, truncating high-weight Pauli operators introduces a controlled but nonzero error.

The Monte Carlo procedure provides an estimate of the MSE associated with a chosen truncation weight k, thus quantifying the accuracy of the expectation value obtained at this level of approximation. The core idea is to randomly sample the many possible ways an observable O could evolve through the circuit—that is, possible sequences of intermediate states, known as Pauli paths—with the sampling more likely to select those paths that have a greater impact on the final result.

Figure 2 visualizes the sampling process in a quantum circuit of many layers. Starting from O as a single Pauli operator, backward propagation through each layer expands it into a linear combination of possible operators. At each layer, one operator is randomly sampled (purple), while the rest are discarded (gray). This procedure yields a Pauli path $(s_0, s_1, \ldots, s_L = O)$ tracing the observable's evolution; repeated sampling of such paths, together with the initial state, certifies the simulation error.

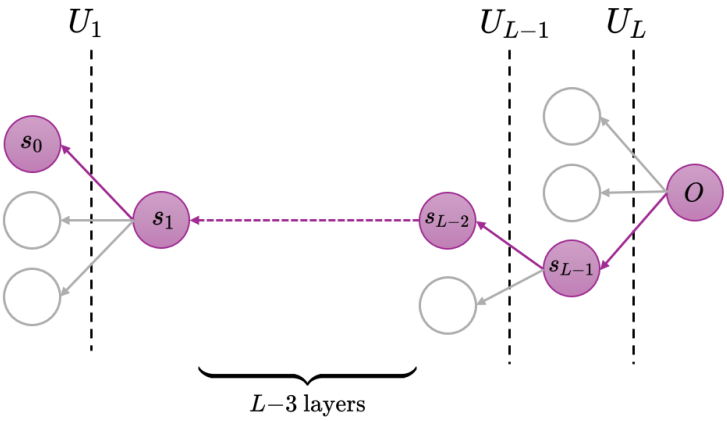


Figure 2: Schematic of Monte Carlo sampling. Each U_j $(j = 1, 2, \ldots, L)$ is a circuit layer, meaning a set of gates on the qubits.

Classical Simulation Baselines

Two baselines are used for benchmarking Pauli propagation in this work: statevector and tensor network methods. Statevector simulation is exact but memory-intensive; tensor network methods handle low-entanglement circuits efficiently but fail for high entanglement.

3 Key Results

3.1 Staircase Random $SU(4)$ Circuit

To benchmark Pauli propagation in a genuinely hard setting, a 36-qubit staircase circuit (Fig. 3) with multiple repetitions was simulated, where each repetition applies 60 random SU(4) gates (i.e., fully random two-qubit operations) that rapidly spread information across the system. Such circuits defeat even state-of-the-art tensor network simulation [2].

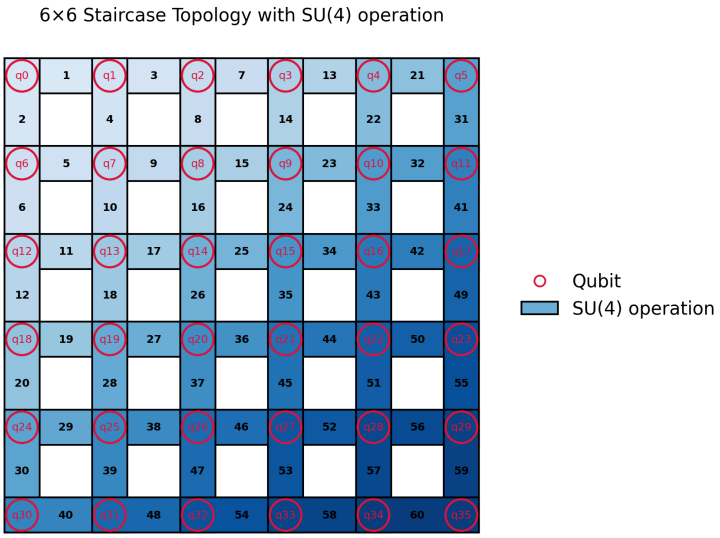
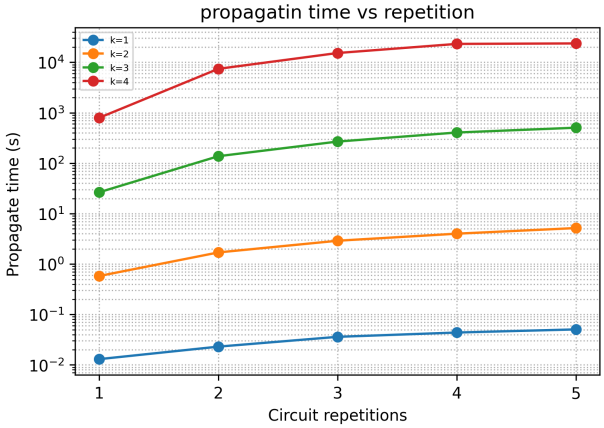
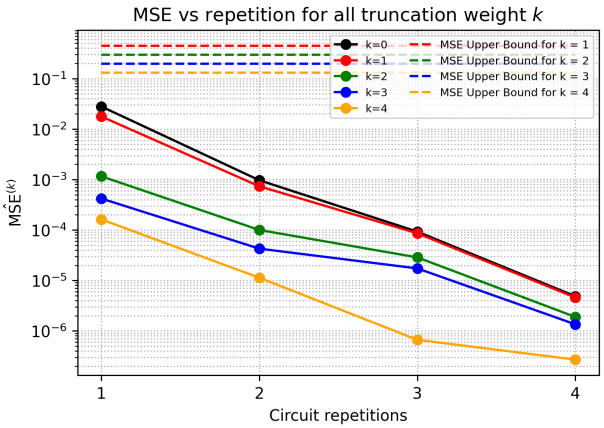


Figure 3: Schematic of the 6×6 staircase topology for a single repetition (contains 60 gates). Each blue rectangle represents a random SU(4) quantum gate, with the number inside indicating the placement order: lighter blue corresponds to earlier gates. Crimson circles denote qubits, with their indices shown inside each circle.

With observable set to Z_{35} (the Z observable on qubit 35), runtime grows only polynomially with repetition but exponentially with truncation weight k (Fig. 4a); even at $k = 4$, four repetitions remain tractable on a workstation, with simulation completing in ∼6 hours, while both exact state-vector and tensor network methods are already infeasible. Monte Carlo certification shows the truncation MSE drops rapidly with increasing k and remains well below the theoretical upper bound (Fig. 4b).



(a) Pauli propagation runtime versus circuit repetition for truncation weights $k = 1, 2, 3, 4$.



(b) Estimated truncation MSE versus circuit repetitions for various k. Dashed lines show the analytic upper bound.

Figure 4: Experimental results on a 36-qubit random staircase circuit

These results confirm that Pauli propagation, equipped with error analysis, extends classical simulation far beyond the reach of tensor-network methods while retaining quantitative reliability.

3.2 Kicked Transverse-Field Ising Circuit

The algorithm was benchmarked against experimental data from IBM's 127-qubit Eagle quantum processor [3], whose topology is shown in Fig. 5. The experiment implements kicked Ising dynamics: starting from an initial state, the system evolves under a cycle of entangling and single-qubit operations, repeated 20 times. The expectation value $\langle Z_{62} \rangle$ is then measured.

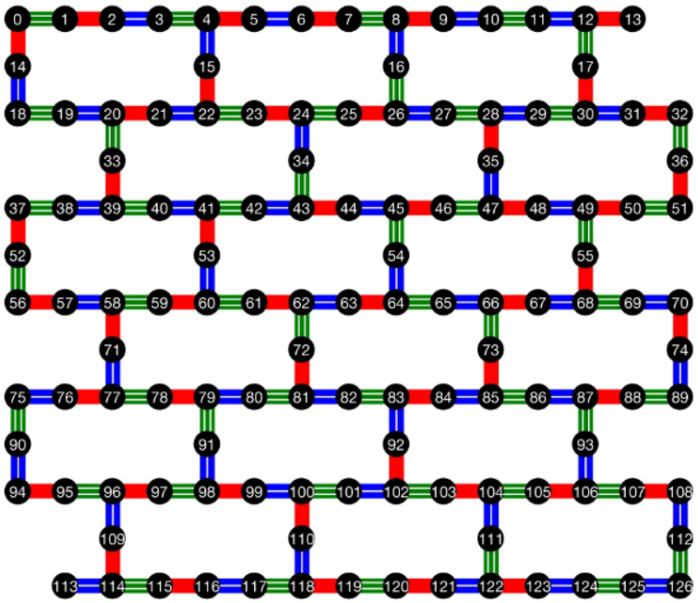


Figure 5: Topology of IBM's 127-qubit Eagle processor. Each node is a qubit, and edges indicate nearest-neighbour connectivity. Adapted from [3]

As illustrated in Fig. 6, Pauli propagation at $k = 7$ completed in only 20 minutes for each experimental parameter θ_h (Intel i7-13700H, serial), agreeing with Eagle data within uncertainty at nearly all points. This is about **90 times** faster than the 30-hour-per-point tensor network calculation on a 64-core server [3], demonstrating a clear advance in classical simulation for regimes previously considered intractable, though recent belief-propagation tensor network methods have also achieved comparable results for this specific architecture [4]. Pauli propagation, however, applies broadly to arbitrary circuit layouts without requiring lattice-specific assumptions.

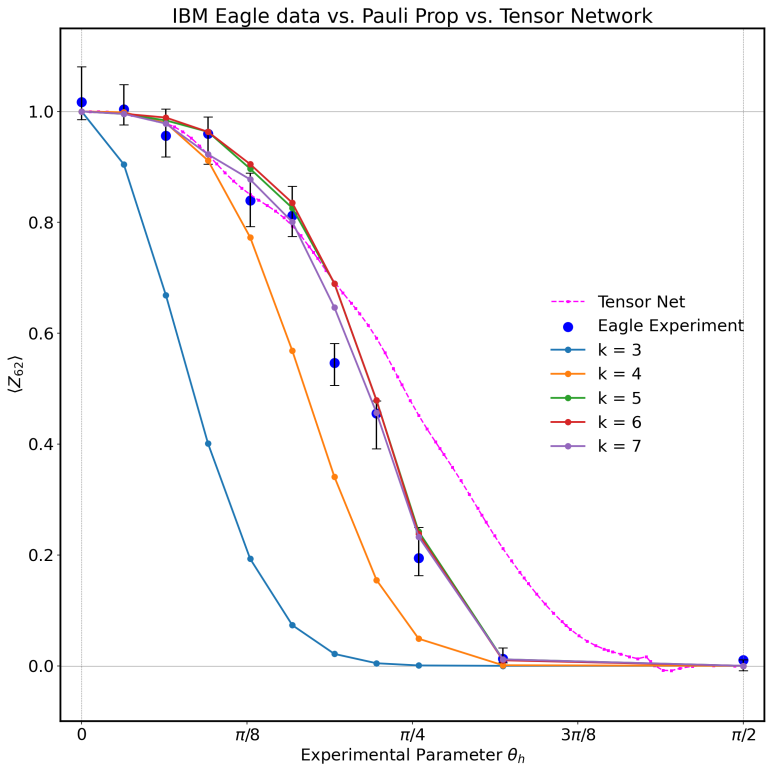


Figure 6: Comparison of Eagle experiment, tensor network, and Pauli propagation estimates of the expectation value $\langle Z_{62} \rangle$ as a function of θ_h after 20 cycles. Eagle results are error-mitigated (blue dots, with error bars), tensor network is shown in magenta dashed, and Pauli propagation results are given for truncation weights $k = 3$–7. Eagle and tensor network data from [5].

4 Conclusion and Outlook

This project demonstrates that Pauli propagation enables simulation of quantum circuits beyond the reach of conventional statevector and tensor network methods. By truncating high-weight Pauli operators, the approach achieves rapid, accurate estimation of expectation values in large, deep circuits.

While BP-TNS approaches can achieve results comparable to Pauli propagation for specific connectivities such as the heavy-hex lattice, Pauli propagation stands out for its generality and minimal structural assumptions.

Classical simulation remains fundamental for benchmarking quantum advantage: only with a strong classical baseline can quantum hardware be justified. Pauli propagation elevates this baseline and reduces dependence on costly quantum resources by enabling fast, offline parameter tuning for quantum variational algorithms. Future integration with noise models will enable more realistic, device-specific simulation, further accelerating hardware development and algorithm design. As a scalable, validated tool, Pauli propagation is poised to become an essential component of the quantum computing ecosystem.

References

[1] A. Angrisani, A. Schmidhuber, M. S. Rudolph, M. Cerezo, Z. Holmes, and H.-Y. Huang, *Classically estimating observables of noiseless quantum circuits*, arXiv:2409.01915 (2024).

[2] H.-K. Zhang, S. Liu, and S.-X. Zhang, *Absence of barren plateaus in finite local-depth circuits with long-range entanglement*, Phys. Rev. Lett. **132**, 150603 (2024).

[3] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. van den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, and A. Kandala, *Evidence for the utility of quantum computing before fault tolerance*, Nature **618**, 500–505 (2023).

[4] J. Tindall, M. Fishman, E. M. Stoudenmire, and D. Sels, *Efficient tensor network simulation of IBM's Eagle kicked Ising experiment*, PRX Quantum **5**, 010308 (2024).

[5] Y. Kim, *Evidence for the utility of quantum computing before fault tolerance (code repository)*, GitHub, `https://github.com/youngseok-kim1/Evidence-for-the-utility-of-quantum-computing-before-fault-tolerance`, accessed July 2, 2025.

Appendix: Declaration of AI Usage

ChatGPT assisted with code generation from pseudocode, English grammar correction, and LaTeX formatting.