Classical Simulation of Quantum Circuits with Pauli Propagation

Steven Ma (ym432)
Supervised by Prof. Hamza Fawzi

July 2025

Word Count: 6859

Contents

1 Introduction

Quantum algorithms running on quantum hardware are theoretically capable of offering super-polynomial speed-ups over classical techniques for Hamiltonian simulation [1], as native superposition and entanglement enable time-evolution to be encoded in polynomial-depth circuits, whereas classical methods require exponentially large matrices. However, quantum hardware is only necessary if it can surpass the most advanced classical simulation algorithms; the bar for quantum advantage rises whenever the classical frontier advances. Pauli propagation, recently proposed by Angrisani *et al.* [2], raises this frontier by evolving observables in the Heisenberg picture and truncating exponentially-suppressed high-weight Pauli strings, extending classical reach beyond tensor-network methods, which rely on low entanglement and fail for deep, scrambling circuits. Such a high-performance simulator is indispensable for validating near-term quantum hardware and quantifying future speed-up. This work presents a rigorously tested implementation of Pauli propagation and applies it to random $SU(4)$ staircase circuits—with Monte Carlo mean squared error (MSE) estimation—and to the kicked Ising experiment on the IBM-Eagle quantum processor, providing a stringent baseline for forthcoming quantum devices.

2 Theory and Background

Definition 1 (Unitary and Special Unitary Matrices)**.** *For an $n \times n$ complex matrix U, if*

$$U^\dagger U = UU^\dagger = I, \tag{2.1}$$

*then U is called a **unitary matrix**. In addition, if $\det(U) = 1$, then U is called a **special unitary matrix**. The sets of all such matrices are denoted by $U(n)$ and $SU(n)$, respectively.*

Unitary matrices describe physically allowed quantum evolutions, while $SU(n)$ removes irrelevant global phases, retaining only physically meaningful operations.

Unitary matrices represent all physically allowed quantum evolutions in closed (i.e., isolated and noiseless) quantum systems.

2.1 The Schrödinger Picture

Bra–Ket Notation

The quantum state $|\psi\rangle$ evolves dynamically under the Hamiltonian or quantum circuit, while observables O remain fixed.

The state at time t is represented by a vector $|\psi(t)\rangle$ in a Hilbert space (i.e., the 2^n-dimensional complex vector space $(\mathbb{C}^2)^{\otimes n}$). Its evolution is governed by the time-dependent Schrödinger equation:

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = H(t) |\psi(t)\rangle, \tag{2.2}$$

where $H(t)$ is the (possibly time-dependent) Hamiltonian. Expectation values of observables O are then computed as

$$\langle O \rangle_t = \langle \psi(t)| O |\psi(t)\rangle. \tag{2.3}$$

In the case of quantum circuits, $U = U_L U_{L-1} \dots U_1$ is typically a product of discrete unitary gates. The final state is

$$|\psi_{\text{final}}\rangle = U_L U_{L-1} \dots U_1 |\psi_{\text{init}}\rangle = U |\psi_{\text{init}}\rangle. \tag{2.4}$$

The expectation value of an observable O is then

$$f_U(O) = \langle \psi_{\text{final}}| O |\psi_{\text{final}}\rangle. \tag{2.5}$$

Density matrix formalism

All the above expressions can be equivalently written in matrix notation. The pure quantum state can be represented as a density matrix $\rho(t) = |\psi(t)\rangle \langle \psi(t)|$, the expectation value of O is given by the trace formula:

$$\langle O \rangle_t = \mathrm{Tr}\big[\, \rho(t) O \,\big] = \langle \psi(t)| O |\psi(t)\rangle. \tag{2.6}$$

The evolution of the state under a unitary U is then:

$$\rho_{\text{final}} = U \rho_{\text{init}} U^\dagger. \tag{2.7}$$

For a quantum circuit, expectation becomes:

$$f_U(O) = \operatorname{Tr}\big[\, U \rho_{\text{init}} U^\dagger O \,\big]. \tag{2.8}$$

2.2 The Heisenberg Picture

Bra–Ket Notation

In the Heisenberg picture, the quantum state $|\psi\rangle$ remains fixed, while observables O evolve dynamically under the action of the Hamiltonian $H(t)$ or quantum circuit U.

The state is time-independent and represented by $|\psi\rangle$ in a Hilbert space. The time evolution is instead transferred to observables, governed by the Heisenberg equation of motion:

$$i\hbar \frac{d}{dt} O_H(t) = [O_H(t), H(t)], \tag{2.9}$$

where $O_H(t)$ denotes the observable in the Heisenberg picture, and $[\cdot,\cdot]$ is the commutator.

Expectation values are then computed as

$$\langle O \rangle_t = \langle \psi |\, O_H(t)\, | \psi \rangle . \tag{2.10}$$

For quantum circuits, the Heisenberg-evolved observable under a unitary U (a product of gates) is

$$O_H = U^\dagger O U, \tag{2.11}$$

and the expectation value becomes

$$f_U(O) = \langle \psi_{\text{init}} |\, U^\dagger O U \,| \psi_{\text{init}} \rangle . \tag{2.12}$$

Density matrix formalism

All the above expressions can be equivalently written using the density matrix. The state remains fixed as ρ_{init}, and the observable evolves. The expectation value is given by:

$$\langle O \rangle_t = \operatorname{Tr}\big[\, \rho_{\text{init}}\, O_H(t) \,\big] = \langle \psi |\, O_H(t)\, | \psi \rangle . \tag{2.13}$$

The evolution of the observable under a unitary U is then:

$$O_H = U^\dagger O U. \tag{2.14}$$

Thus, for a quantum circuit, the expectation is

$$f_U(O) = \operatorname{Tr}\big[\, U^\dagger O U \, \rho_{\text{init}} \big]. \tag{2.15}$$

This matrix form is especially useful for generalizing to mixed states and for later connection to the Heisenberg picture and Pauli propagation methods.

Equivalence. Heisenberg and Schrödinger pictures yield identical physical predictions. For any observable O and initial state $|\psi\rangle$,

$$\langle \psi |\, U^\dagger O \, U \,| \psi \rangle = \langle \psi' |\, O \,| \psi' \rangle , \quad \text{where } |\psi'\rangle = U \,|\psi\rangle . \tag{2.16}$$

The time evolution can thus be equivalently viewed as acting on observables or on quantum states.

2.3 Pauli Operators and Pauli Matrices

The Pauli operators $\{I, X, Y, Z\}$ are a fundamental set of Hermitian, unitary matrices acting on a single qubit. The Pauli matrices are defined as

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The X operator exchanges $|0\rangle$ and $|1\rangle$, acting like a classical NOT gate. The Z operator leaves $|0\rangle$ unchanged and flips the sign of $|1\rangle$. The Y operator combines a bit-flip and a phase-flip, mapping $|0\rangle$ to $i\,|1\rangle$ and $|1\rangle$ to $-i\,|0\rangle$. The eigenvalues and eigenstates of the Pauli matrices are summarized in Table 1.

Pauli Matrix	Eigenvalue	Eigenvector
Z	$+1$	$\|0\rangle$
Z	-1	$\|1\rangle$
X	$+1$	$\|+\rangle = \frac{\|0\rangle + \|1\rangle}{\sqrt{2}}$
X	-1	$\|-\rangle = \frac{\|0\rangle - \|1\rangle}{\sqrt{2}}$
Y	$+1$	$\|r\rangle = \frac{\|0\rangle + i\,\|1\rangle}{\sqrt{2}}$
Y	-1	$\|l\rangle = \frac{\|0\rangle - i\,\|1\rangle}{\sqrt{2}}$

Table 1: Eigenvalues and eigenstates of the Pauli matrices.

On n qubits, tensor products of Pauli matrices form a complete orthonormal basis for all operators. Any operator O can be expanded as

$$O = \sum_{P \in \{I,X,Y,Z\}^{\otimes n}} a_P P, \qquad a_P = \frac{1}{2^n} \mathrm{Tr}[OP], \tag{2.17}$$

where each P is a tensor product of single-qubit Pauli operators and each coefficient $a_P \in \mathbb{C}$.

The orthonormality relation $\mathrm{Tr}[PQ] = 2^n \delta_{P,Q}$ for Pauli strings P, Q ensures that this decomposition is unique.

Terminology.

- **Pauli operator:** One of the single-qubit matrices I, X, Y, Z.
- **Pauli string:** A tensor product of Pauli operators on n qubits, i.e., $P \in \{I, X, Y, Z\}^{\otimes n}$.
- **Pauli term:** A Pauli string multiplied by a complex scalar coefficient, i.e., $a_P P$.
- **Pauli series:** A finite linear combination of Pauli terms, i.e., $\sum_P a_P P$.

3 Other Quantum Circuit Simulation Algorithms

3.1 Statevector Simulation

Statevector simulation is the most direct approach for classically simulating quantum circuits and operates in the Schrödinger picture.

For an n-qubit system, the quantum state is represented by a complex vector of size 2^n:

$$|\psi\rangle \in \mathbb{C}^{2^n}. \tag{3.1}$$

Each gate U_j in the circuit corresponds to a $2^n \times 2^n$ unitary matrix acting on the full statevector. The overall evolution for a depth-L circuit is given by Equation 2.4 and expectation value of an observable O is computed by Equation 2.5. The space complexity is $O(2^n)$, as the full statevector (a complex array of size 2^n) is stored in memory.

While statevector simulation is exact and conceptually simple, the exponential scaling of memory limits this approach to circuits with $n \lesssim 35$ qubits on classical hardware.

3.2 Tensor Network Simulation

Tensor network simulation is a powerful method for classically simulating quantum circuits with limited entanglement, operating in the Schrödinger picture.

A quantum state of n qubits is represented as a network of low-rank tensors, typically arranged as a matrix product state (MPS). For an MPS, the state is factorized as:

$$|\psi\rangle = \sum_{i_1,\dots,i_n} A^{[1]}_{i_1} A^{[2]}_{i_2} \cdots A^{[n]}_{i_n} |i_1 \dots i_n\rangle , \tag{3.2}$$

where each $A^{[k]}_{i_k}$ is a matrix of bond dimension χ.

Quantum gates are applied locally by updating the relevant tensors, with singular value decomposition used to maintain the network's structure and truncate negligible entanglement. The memory footprint of an MPS scales as $O(n\,\chi^2)$ [3].

Tensor network methods exploit the area law of entanglement to efficiently simulate low-entanglement circuits. However, for highly entangling or random circuits, the required bond dimension χ grows rapidly, and the simulation becomes intractable.

In practice, tensor network simulation enables exact or approximate treatment of circuits with $n \sim 50$–100 qubits, provided the entanglement remains sufficiently bounded.

4 Pauli Propagation Algorithm

4.1 Pauli Operator Decomposition and Heisenberg Evolution

Estimating expectation values of observables in large quantum circuits is classically challenging due to the exponential growth of Hilbert space with the number of qubits. Pauli propagation methods address this challenge by expressing the observable O as a linear combination of Pauli operators $O = \sum_{P\in\{I,X,Y,Z\}^{\otimes n}} a_P P$ (see Equation 2.17).

In the Heisenberg picture, the observable O is evolved backwards through the circuit $U = U_L U_{L-1} \cdots U_1$ by conjugation under each circuit layer. Here, a layer refers to a set of gates that act in parallel on disjoint sets of qubits and can be applied simultaneously in a single time step. Stacking multiple layers sequentially yields a circuit of depth L. The final propagated observable O_U is given by

$$O_U = U^\dagger O U = U_1^\dagger \cdots U_L^\dagger O U_L \cdots U_1. \tag{4.1}$$

4.2 Exponential Complexity and the Need for Truncation

However, in general circuits, each non-Clifford gate transforms a single Pauli term into a linear combination of many Pauli terms, so the number of terms in the Pauli series grows exponentially with circuit depth. This exponential blowup makes exact Pauli propagation intractable for large systems.

To address this, a truncation scheme is employed: at each step, only low-weight Pauli terms are retained, where the weight of a Pauli string P, denoted $|P|$, is defined as the number of non-identity single-qubit Pauli operators in P (e.g., $Y_2 \otimes I_1 \otimes X_0$ has weight 2). By discarding high-weight terms, the computational cost remains manageable, at the expense of a controlled approximation error. This low-weight truncation enables classically efficient estimation of expectation values in noiseless quantum circuits for sufficiently large truncation weight.

4.3 Pauli propagation Algorithm

The Pauli propagation algorithm proceeds as follows [2]:

1. **Low-weight truncation of the observable.**
   Given the observable $O = \sum_{P\in\{I,X,Y,Z\}^{\otimes n}} a_P P$, construct its low-weight approximation by retaining only Pauli strings of weight at most k:

   $$O_L^{(k)} = \sum_{P:|P|\le k} a_P P, \tag{4.2}$$

   where the superscript (k) indicates truncation to Pauli strings of weight $\le k$.

2. **Iterative Heisenberg propagation with truncation.**
   For each layer $j = L, L-1, \ldots, 2$:

   (a) Back-propagate the observable through the layer (without truncation):

   $$\tilde{O}_{j-1} = U_j^\dagger O_j^{(k)} U_j. \tag{4.3}$$

   (b) Truncate the propagated observable to weight k:

   $$O_{j-1}^{(k)} = \mathcal{T}_k\left(\tilde{O}_{j-1}\right) = \sum_{P:|P|\le k} \frac{1}{2^n} \mathrm{Tr}\left[\tilde{O}_{j-1}P\right] P. \tag{4.4}$$

   where $\mathcal{T}_k$ denotes the projection onto Pauli strings of weight $\le k$ (the resulting operator is therefore a Pauli series supported on those strings).

3. **Final observable and expectation value.**
   After propagating through all layers, apply the first layer and obtain the truncated observable

   $$O_U^{(k)} = U_1^\dagger O_1^{(k)} U_1 = \sum_{P:|P|\le k} \frac{1}{2^n} \mathrm{Tr}\left[U_1^\dagger O_1^{(k)} U_1\, P\right] P \tag{4.5}$$

   The expectation value obtained by truncating all Pauli strings with weight greater than k is

   $$\tilde{f}_U^{(k)}(O) = \mathrm{Tr}\left[\, O_U^{(k)}\, \rho_{\mathrm{init}}\right]. \tag{4.6}$$

This truncation-based propagation efficiently controls the growth of Pauli terms, allowing scalable simulation of quantum circuits with a rigorously bounded approximation error. Intuitively, the coefficients of high-weight Pauli operators are suppressed during back-propagation through the circuit, causing their contributions to expectation values to become small.

Figure 1 schematically illustrates truncated Pauli propagation (with truncation weight $k = 2$). The observable is back-propagated from the L-th circuit layer. After the first non-Clifford gate, it becomes a Pauli series with terms of weights 2 and 1. At the $(L-1)$-th non-Clifford layer, each term further branches, yielding four terms in total. After the $(L-2)$-th non-Clifford layer, one term splits into two, one of which has weight 3 and is truncated; only terms with weight at most k continue to propagate.

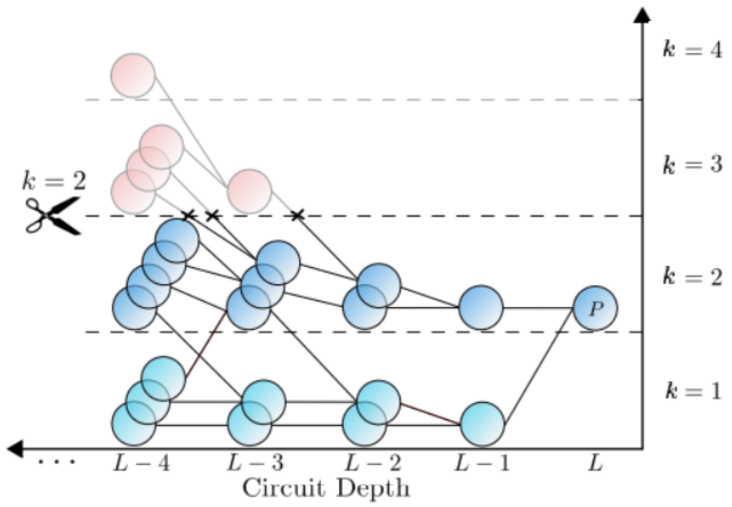


Figure 1: Schematic depiction of Pauli propagation with weight truncation ($k = 2$). Adapted from [2].

4.4 Theoretical Guarantees of Pauli propagation

Note: The theorems in this report are adapted from [2, Theorems 1, 2, and 4], and the definitions are likewise adapted from the same reference for clarity and completeness in this context.

The following theorems are stated under the assumption that the circuit $U = U_L U_{L-1} \cdots U_1$ consists of L layers, with each layer U_j independently sampled from a locally scrambling distribution. Specifically, a locally scrambling distribution is invariant under conjugation by any tensor product of single-qubit Clifford gates.

Definition 2 (Single-qubit Clifford Gate). *[4]*
If a unitary operator C acts on a single qubit and, for every Pauli operator $P \in \{X, Y, Z\}$,

$$CPC^\dagger \in \{\pm X, \pm Y, \pm Z\}, \quad \text{and equivalently,} \quad C^\dagger PC \in \{\pm X, \pm Y, \pm Z\},$$

*then C is called a **single-qubit Clifford gate**. Both $CPC^\dagger$ and $C^\dagger PC$ are Pauli operators (up to sign), although they are not generally equal.*

Definition 3 (Locally Scrambling Distribution). *[2] Let $\mathcal{D}$ be a probability distribution over n-qubit unitaries. If, for every Clifford string $C = C_1 \otimes \cdots \otimes C_n$ and for all Hermitian operators H, H', the following holds:*

$$\mathbb{E}_{U_j \sim \mathcal{D}} \left[U_j^{\otimes 2} (H \otimes H') U_j^{\dagger \otimes 2} \right] = \mathbb{E}_{U_j \sim \mathcal{D}} \left[(CU_j)^{\otimes 2} (H \otimes H') (CU_j)^{\dagger \otimes 2} \right],$$

*that is, the distribution $\mathcal{D}$ is invariant up to the second moment under conjugation by any product of single-qubit Clifford gates, then $\mathcal{D}$ is called a **locally scrambling distribution** (and $U_j \sim \mathcal{D}$ is a unitary sampled from a locally scrambling distribution).*

Definition 4 (L-layered Locally Scrambling Circuit Ensemble). *[2] An **L-layered locally scrambling circuit ensemble** is a probability distribution over n-qubit quantum circuits of the form*

$$U = U_L U_{L-1} \cdots U_1,$$

where each layer U_j is independently sampled from a locally scrambling distribution $\mathcal{D}$ over n-qubit unitaries, and each U_j can be decomposed as a product of local gates, each acting on a small subset of qubits.

Intuitively, a locally scrambling distribution means that the action of each circuit layer rapidly disperses any local (low-weight) Pauli string into a superposition of higher-weight Pauli strings.

Theorem 4.1 (Mean Squared Error Bound). *The MSE is used as the primary performance measure:*

$$\mathbb{E}_U \left[(\Delta f_U^{(k)}(O))^2 \right] = \mathbb{E}_U \left[\left(f_U(O) - \tilde{f}_U^{(k)}(O) \right)^2 \right], \tag{4.7}$$

where the expectation is taken over circuits U sampled from a locally scrambling circuit ensemble.
For any Pauli weight $k \geq 0$, the following bound holds:

$$\mathbb{E}_U \left[(\Delta f_U^{(k)}(O))^2 \right] \leq \left(\frac{2}{3} \right)^{k+1} \|O\|_{\text{Pauli},2}^2, \tag{4.8}$$

where $\|O\|_{\text{Pauli},2} = \left(2^{-n} \text{Tr}[O^\dagger O] \right)^{1/2}$ is the normalized Hilbert-Schmidt norm.

Remarks.

- Given an observable $O = \sum_P a_P P$ expressed in the Pauli basis [2],

$$\|O\|_{\text{Pauli},2}^2 = \sum_P a_P^2. \tag{4.9}$$

- If all eigenvalues of O lie in $[-1, 1]$, the MSE decays exponentially with the truncation weight k:

$$\mathbb{E}_U \left[(\Delta f_U^{(k)}(O))^2 \right] \leq \left(\frac{2}{3} \right)^{k+1}. \tag{4.10}$$

- Applying Markov's inequality gives a probabilistic error bound:

$$\Pr_U \left[|f_U(O) - \tilde{f}_U^{(k)}(O)| \leq \epsilon \|O\|_{\text{Pauli},2} \right] \geq 1 - \delta, \quad k \in \Omega\left(\log(\epsilon^{-1}\delta^{-1}) \right) \tag{4.11}$$

  That is, to ensure with probability at least $1-\delta$ that the estimation error does not exceed ϵ, the truncation weight k must scale at least as $\log(\epsilon^{-1}\delta^{-1})$.

Theorem 4.2 (Time Complexity)**.** *Let U be a quantum circuit drawn from an L-layered locally scrambling circuit ensemble on n qubits, and let O be any observable. Then, for any error $\epsilon > 0$ and failure probability $\delta > 0$, the Pauli propagation algorithm with truncation weight $k = O(\log(\epsilon^{-1}\delta^{-1}))$ has the following guarantees:*

- *The algorithm runs in time $L\, n^{O(\log(\epsilon^{-1}\delta^{-1}))}$ and outputs α such that*
$$|\alpha - f_U(O)| \leq \epsilon \, \|O\|_{\mathrm{Pauli},2} \tag{4.12}$$
with probability at least $1 - \delta$.
- *If each circuit layer acts locally on at most D qubits, then the runtime improves to $L\, O(D \log(\epsilon^{-1}\delta^{-1}))$.*

Thus, for any fixed error and failure probability, the classical simulation requires only polynomial resources.

Exponential decay of Pauli expectation values.

For a n-qubit circuit U sampled from a locally scrambling circuit ensemble, and any $P \in \{I, X, Y, Z\}^{\otimes n}$, the following statistical property holds [2]: for any initial state ρ_{init},

$$\mathbb{E}_U \left[\left(\mathrm{Tr}[U^\dagger P U \, \rho_{\mathrm{init}}] \right)^2 \right] \leq (2/3)^{|P|}. \tag{4.13}$$

By Jensen's inequality [5], it follows that

$$\mathbb{E}_U \left[\left| \mathrm{Tr}[U^\dagger P U \, \rho_{\mathrm{init}}] \right| \right] \leq (2/3)^{|P|/2}. \tag{4.14}$$

This shows that high-weight Pauli observables are exponentially suppressed after propagation through such random circuits. Note that this is a loose upper bound, but it quantitatively justifies the validity of the high-weight truncation scheme.

5 Error Estimate with Monte Carlo Sampling

Theorem 4.1 provides an upper bound on the MSE averaged over random circuits from a locally scrambling circuit ensemble, but this bound is typically loose for concrete circuit instances.

To estimate the MSE introduced by weight truncation in Pauli propagation for a specific quantum circuit U, a Monte Carlo sampling approach is employed [2]. This method samples random Pauli paths according to the relative amplitude of each pauli term defined by Heisenberg back-propagation of the observable.

Given a particular quantum circuit $U = U_L U_{L-1} \ldots U_1$ of depth L, and an observable O, first expand O as a Pauli series:
$$O = \sum_{P \in \mathcal{P}_n} a_P P, \qquad a_P = \frac{1}{2^n} \mathrm{Tr}[OP], \tag{5.1}$$

where $\mathcal{P}_n = \{I, X, Y, Z\}^{\otimes n}$ is the n-qubit Pauli group.

After Heisenberg evolution (without truncation) through the entire circuit $U = U_L \cdots U_1$, the observable O transforms as
$$U^\dagger O U = \sum_{\gamma_i \in \Gamma} \Phi_{\gamma_i}(U) \, s_0^{(\gamma_i)}, \tag{5.2}$$

Each term $s_0^{(\gamma_i)}$ in the sum is the terminating Pauli string of a Pauli path γ_i, where a Pauli path is defined as an ordered sequence of Pauli strings:
$$\gamma_i = (s_0^{(\gamma_i)}, \, s_1^{(\gamma_i)}, \, \ldots, \, s_L^{(\gamma_i)}).$$

A Pauli path is sampled by starting from the observable O and, at each layer, performing a backward Heisenberg propagation to expand into a Pauli series, then selecting one specific Pauli string from this series at each step. Within each path, $s_0^{(\gamma_i)}$ is the Pauli string obtained after propagating backward through all layers, and is the operator that ultimately acts on the initial state. Γ denotes the set of all possible Pauli paths, and $\Phi_{\gamma_i}(U)$ is the amplitude associated with path γ_i (Both the path sampling procedure and the explicit formula for Φ_{γ_i} are detailed below).

Let $\Gamma_{\leq k}$ denote the set of all Pauli paths $\gamma_i \in \Gamma$ whose maximal Pauli weight across all layers does not exceed k, and let $\Gamma_{>k} = \Gamma \setminus \Gamma_{\leq k}$ denote its complement. For a given initial state ρ_{init}, the truncation error is

$$\begin{aligned}\Delta f_U^{(k)}(O) &= f_U(O) - \tilde{f}_U^{(k)}(O) = \mathrm{Tr}[U^\dagger O U \, \rho_{\text{init}}] - \mathrm{Tr}[O_U^{(k)} \, \rho_{\text{init}}] \\ &= \sum_{\gamma_i \in \Gamma} \Phi_{\gamma_i}(U) \, d_{\gamma_i} - \sum_{\gamma_i \in \Gamma_{\leq k}} \Phi_{\gamma_i}(U) \, d_{\gamma_i} = \sum_{\gamma_i \in \Gamma_{>k}} \Phi_{\gamma_i}(U) \, d_{\gamma_i}\end{aligned} \tag{5.3}$$

where $d_{\gamma_i} = \mathrm{Tr}\left[s_0^{(\gamma_i)} \, \rho_{\text{init}}\right]$ encodes the overlap with the initial state. For $\rho_{\text{init}} = |0\rangle\langle 0|^{\otimes n}$, one has $d_{\gamma_i} = 1$ if $s_0^{(\gamma_i)} \in \{I, Z\}^{\otimes n}$, and $d_{\gamma_i} = 0$ otherwise.

The corresponding mean squared error is

$$\mathbb{E}_U\left[\left(\Delta f_U^{(k)}(O)\right)^2\right] = \mathbb{E}_U\left[\left(\sum_{\gamma_i \in \Gamma_{>k}} \Phi_{\gamma_i}(U) \, d_{\gamma_i}\right)^2\right]. \tag{5.4}$$

Due to orthogonality of Pauli paths [2], their amplitudes are uncorrelated:

$$\mathbb{E}_U[\Phi_\gamma(U) \, \Phi_{\gamma'}(U)] = 0, \quad \text{for } \gamma \neq \gamma'. \tag{5.5}$$

which yields

$$\mathbb{E}_U\left[\left(\Delta f_U^{(k)}(O)\right)^2\right] = \sum_{\gamma \in \Gamma_{>k}} \Phi_\gamma(U)^2 \, d_\gamma^2. \tag{5.6}$$

To estimate Equation. (5.6) unbiasedly, Monte Carlo sampling is required, as the full sample space $\Gamma_{>k}$ is prohibitively large.

5.1 Sampling Algorithm

Repeat the following steps for $i = 1, \ldots, N$ to sample N independent Pauli paths:

1. **Initial sampling.**
   Sample a Pauli string $s_L^{(\gamma_i)} \in \mathcal{P}_n$ from the distribution

$$\Pr\big[s_L^{(\gamma_i)} = P\big] = \frac{a_P^2}{\sum_Q a_Q^2} = \frac{\mathrm{Tr}[OP]^2}{\|O\|_{\text{Pauli},2}^2}. \tag{5.7}$$

2. **Iterative back-propagation.**
   For $j = L, L-1, \ldots, 1$, propagate the Pauli string $s_j^{(\gamma_i)}$ backward through layer U_j:

   (a) Compute the Heisenberg conjugation:

$$U_j^\dagger s_j^{(\gamma_i)} U_j = \sum_{t \in \mathcal{P}_n} c_{j \to t} \, t, \qquad c_{j \to t} = \frac{1}{2^n} \mathrm{Tr}\big[U_j^\dagger s_j^{(\gamma_i)} U_j \, t\big], \tag{5.8}$$

   (b) Sample the next Pauli string $s_{j-1}^{(\gamma_i)} \in \mathcal{P}_n$ according to

$$\Pr\big[s_{j-1}^{(\gamma_i)} \mid s_j^{(\gamma_i)}\big] = \frac{\big|c_{j \to s_{j-1}^{(\gamma_i)}}\big|^2}{\sum_{t \in \mathcal{P}_n} \big|c_{j \to t}\big|^2}. \tag{5.9}$$

3. **Pauli path generation and Monte-Carlo sampling.**
   Steps 1–2 yield a Pauli path $\gamma_i = \big(s_0^{(\gamma_i)}, s_1^{(\gamma_i)}, \ldots, s_L^{(\gamma_i)}\big)$ with amplitude

$$\Phi_{\gamma_i}(U) = a_{s_L^{(\gamma_i)}} \prod_{j=1}^{L} c_{j \to s_{j-1}^{(\gamma_i)}} = \frac{1}{2^n} \mathrm{Tr}\big[O \, s_L^{(\gamma_i)}\big] \prod_{j=1}^{L} \frac{1}{2^n} \mathrm{Tr}\big[U_j^\dagger s_j^{(\gamma_i)} U_j \, s_{j-1}^{(\gamma_i)}\big], \tag{5.10}$$

   which quantifies the total contribution of the path γ_i to the observable's Heisenberg evolution.

This yields N independent Pauli paths $\{\gamma_i\}_{i=1}^N \subset \Gamma$ and their corresponding amplitudes $\{\Phi_{\gamma_i}(U)\}_{i=1}^N$. The sampled paths are distributed according to

$$\pi(\gamma) = \frac{\Phi_\gamma(U)^2}{Z}, \qquad Z = \sum_{\gamma' \in \Gamma} \Phi_{\gamma'}(U)^2. \tag{5.11}$$

Figure. 2 illustrates the Monte Carlo sampling algorithm described above. Suppose the observable O is initially a single Pauli string. After propagating backward through layer L, it expands into a linear combination of three Pauli strings. One of these strings, shown as a purple circle with s_{L-1}, is sampled according to its coefficient, while the discarded Pauli strings are shown in gray. This sampling is repeated at each layer, selecting a single (purple) Pauli string at each step and yielding a trajectory through the Pauli path space that terminates at s_0.

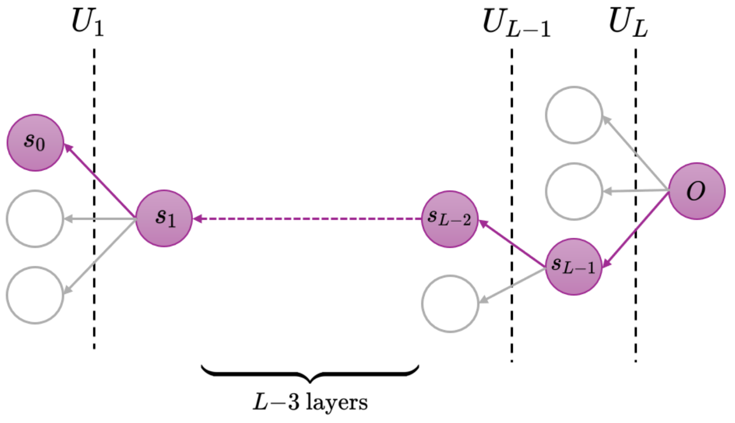


Figure 2: Schematic representation of the Monte Carlo sampling algorithm for Pauli propagation.

5.2 Certified Error Bound for Monte Carlo Estimator

Unbiased MSE estimator

Define

$$X^{(k)}(\gamma) = \begin{cases} Z\, d_\gamma^2, & \gamma \in \Gamma_{>k}, \\ 0, & \gamma \notin \Gamma_{>k}, \end{cases} \tag{5.12}$$

Using the N samples, an unbiased estimator for the MSE in (5.6) is

$$\widehat{\mathrm{MSE}}_U^{(k)} = \frac{1}{N}\sum_{i=1}^N X^{(k)}(\gamma_i), \tag{5.13}$$

Below is a proof of unbiasedness. By linearity and independence,

$$\mathbb{E}_{\{\gamma_i\}\sim\pi}\left[\widehat{\mathrm{MSE}}_U^{(k)}\right] = \mathbb{E}_{\{\gamma_i\}\sim\pi}\left[\frac{1}{N}\sum_{i=1}^N X^{(k)}(\gamma_i)\right] = \frac{1}{N}\sum_{i=1}^N \mathbb{E}_{\gamma_i\sim\pi}\left[X^{(k)}(\gamma_i)\right]. \tag{5.14}$$

Each sample is identically distributed, so

$$\mathbb{E}_{\gamma_i\sim\pi}\left[X^{(k)}(\gamma_i)\right] = \mathbb{E}_{\gamma\sim\pi}\left[X^{(k)}(\gamma)\right] = \sum_{\gamma\in\Gamma} \pi(\gamma) X^{(k)}(\gamma). \tag{5.15}$$

Plugging in the definitions,

$$\sum_{\gamma\in\Gamma} \pi(\gamma) X^{(k)}(\gamma) = \sum_{\gamma\in\Gamma_{>k}} \frac{\Phi_\gamma^2}{Z}(Z\, d_\gamma^2) = \sum_{\gamma\in\Gamma_{>k}} \Phi_\gamma^2\, d_\gamma^2. \tag{5.16}$$

Thus,

$$\mathbb{E}_{\{\gamma_i\}\sim\pi}\left[\widehat{\mathrm{MSE}}_U^{(k)}\right] = \sum_{\gamma\in\Gamma_{>k}} \Phi_\gamma^2\, d_\gamma^2 = \mathbb{E}_U\left[(\Delta f_U^{(k)}(O))^2\right]. \tag{5.17}$$

which matches the right-hand side of (5.6).

Variance of the MSE estimator

By independence and identical distribution of $\{\gamma_i\}_{i=1}^N$,

$$\mathrm{Var}\big[\widehat{\mathrm{MSE}}_U^{(k)}\big] = \mathrm{Var}\Big[\frac{1}{N}\sum_{i=1}^N X^{(k)}(\gamma_i)\Big] = \frac{1}{N^2}\sum_{i=1}^N \mathrm{Var}_{\gamma_i\sim\pi}\big[X^{(k)}(\gamma_i)\big] = \frac{1}{N}\,\mathrm{Var}_{\gamma\sim\pi}\big[X^{(k)}(\gamma)\big]. \tag{5.18}$$

The second moment of $X^{(k)}(\gamma)$ is

$$\mathbb{E}_{\gamma\sim\pi}\big[(X^{(k)}(\gamma))^2\big] = \sum_{\gamma\in\Gamma_{>k}} \frac{\Phi_\gamma^2}{Z}\,(Z\,d_\gamma^2)^2 \;=\; Z\sum_{\gamma\in\Gamma_{>k}} \Phi_\gamma^2 d_\gamma^4, \tag{5.19}$$

and its first moment is given by (5.16). Hence, the theoretical variance of the sample MSE estimator is

$$\mathrm{Var}\big[\widehat{\mathrm{MSE}}_U^{(k)}\big] = \frac{1}{N}\left[Z\sum_{\gamma\in\Gamma_{>k}} \Phi_\gamma^2 d_\gamma^4 - \Big(\sum_{\gamma\in\Gamma_{>k}} \Phi_\gamma^2 d_\gamma^2\Big)^2\right]. \tag{5.20}$$

Sample-based estimator. Let

$$\bar{X}^{(k)} \;=\; \frac{1}{N}\sum_{i=1}^N X^{(k)}(\gamma_i), \qquad S^2 \;=\; \frac{1}{N-1}\sum_{i=1}^N \big(X^{(k)}(\gamma_i) - \bar{X}^{(k)}\big)^2. \tag{5.21}$$

Then $\mathbb{E}_{\{\gamma_i\}\sim\pi}[S^2] = \mathrm{Var}_{\gamma\sim\pi}[X^{(k)}(\gamma)]$, so

$$\widehat{\mathrm{Var}}\big[\widehat{\mathrm{MSE}}_U^{(k)}\big] \;=\; \frac{S^2}{N} = \frac{1}{N(N-1)}\sum_{i=1}^N \big(X^{(k)}(\gamma_i) - \bar{X}^{(k)}\big)^2. \tag{5.22}$$

since S^2 is the unbiased sample MSE variance for independent Pauli paths drawn from π, so Equation (5.22) is an *unbiased* estimator of the theoretical variance in Equation (5.20).

Remark The Pauli-path orthogonality relation (Equation 5.5) is formally guaranteed only for the locally scrambling circuit ensemble [2]. Equation 5.13 is used to estimate the MSE for concrete circuits, acknowledging that this is an approximation.

Theorem 5.1 (Certified error estimate)**.** *Suppose U is independently sampled from an L-layer locally scrambling circuit ensemble. For any observable O, assume that the Pauli-path sampling method described above is used, together with the unbiased MSE estimator. If $N = L\,\varepsilon^{-2}\log(1/\delta)$ independent samples are drawn, then with probability at least $1-\delta$, the output α of the estimator satisfies*

$$\left|\alpha - \mathbb{E}_U\left[\Delta f_U^{(k)}(O)^2\right]\right| \le \varepsilon\,\|O\|_{\mathrm{Pauli},2}^2. \tag{5.23}$$

6 Package Implementation

The Pauli propagation algorithm is implemented as a Python package. The package is integrated with `Qiskit` [6], supporting direct Heisenberg back-propagation of observables through `qiskit.circuit.QuantumCircuit` objects. Internally, Pauli operators are encoded as bit-masks for fast, memory-efficient manipulation, and the implementation covers Heisenberg back-propagation for all standard quantum gates. Both exact propagation (with user-controlled truncation weight k) and Monte Carlo path sampling for MSE estimation are supported.

The `pauli_pkg` package is organized into a modular architecture comprising five core modules: `utils.py` `pauli_term.py`, `gates.py`, `propagator.py`, and `monte_carlo.py`. These are complemented by a unit test suite (validating each gate implementation) and an integration test suite (verifying correctness on full random circuits). Additional modules, including , `decomposition.py`, and `circuit_topologies.py`, are not core components but are provided for experimental support in later sections.

```
pauli_pkg/
|-- pauli_propagation/
|   |-- __init__.py           # Package initialization and exports
|   |-- pauli_term.py         # Core Pauli operator data structure
|   |-- propagator.py         # Main propagation algorithms
|   |-- monte_carlo.py        # Monte Carlo sampling implementation
|   |-- gates.py              # Quantum gate implementations
|   |-- utils.py              # Utility functions and helpers
|   |-- decomposition.py      # SU(4) KAK decomposition
|   |-- circuit_topologies.py # Circuit generation utilities
|-- integration_test/         # Integration test suite
|-- unit_test/                # Unit test suite
|-- pyproject.toml            # Project configuration
```

Figure 3: Directory structure of `pauli_pkg`

6.1 `pauli_term.py`: Bit-Mask Representation of Pauli Operators

The `pauli_term.py` module implements the `PauliTerm` class, the core data structure for representing Pauli terms in `pauli_pkg`. Each `PauliTerm` object corresponds to $a_P P$ and is stored using three fields: a complex coefficient `coeff` representing a_P, an integer bit-mask (`key`) encoding the Pauli string P, and the number of qubits n in the quantum circuit.

A single-qubit Pauli operator is encoded by two bits: (x, z), where $x = 1$ indicates the presence of X and $z = 1$ indicates the presence of Z. The possible combinations for a single qubit are:

(x, z)	Pauli Operator
$(0, 0)$	I
$(1, 0)$	X
$(0, 1)$	Z
$(1, 1)$	Y

Table 2: Mapping between bit mask values (x, z) and single-qubit Pauli operators.

The `PauliTerm` class encodes each Pauli term as a single $2n$-bit integer, where the binary representation of the integer specifies the operator on each qubit: indices start from 0, so for the i-th qubit $(0 \leq i < n)$, the i-th bit of binary is the x mask and the $(i+n)$-th bit is the z mask. The overall integer is `key` $= \sum_{i=0}^{n-1}(x_i\, 2^i + z_i\, 2^{n+i})$.

For example, for $n = 3$, the Pauli term $1.2\, Y_2 \otimes I_1 \otimes X_0$ (using `Qiskit`'s little-endian convention, i.e., Y acts on qubit 2, X on qubit 0) is represented in code as `PauliTerm(1.2, 37, 3)`. The integer 37 in binary is `0b100101`, with each bit interpreted as shown in Table 3:

Bit position	5	4	3	2	1	0
Meaning	z_2	z_1	z_0	x_2	x_1	x_0
Value	1	0	0	1	0	1

Table 3: Bit-mask encoding of `PauliTerm(1.2, 37, 3)` for $n = 3$.

Thus, the bit mask for qubit 0 is $(x_0, z_0) = (1, 0)$, corresponding to X_0; for qubit 1, $(x_1, z_1) = (0, 0)$, corresponding to I_1; and for qubit 2, $(x_2, z_2) = (1, 1)$, corresponding to Y_2.

Encoding both the x and z masks for all n qubits in a single $2n$-bit integer—rather than splitting into two separate integers—reduces memory footprint and computational overhead for moderate n. For most practical cases $(n \leq 64)$, this single-integer approach achieves the best trade-off between simplicity and performance; for very large n, a two-integer scheme may be more suitable.

6.2 `gates.py`: Bitwise Pauli Propagation Rules

The `gates.py` module implements quantum gate propagation rules for the Heisenberg evolution of Pauli operators within the package. Its core is the `QuantumGate` class, which provides a highly extensible gate registry system that allows new quantum gates to be easily added externally via a decorator, without modifying the class itself.

Many quantum gates (e.g., rotation gates) require parameters such as angles θ, which are stored in the `qiskit.circuit.QuantumCircuit` object. The registry allows each gate to be linked with a simple function that extract its parameters from the circuit. When a gate is encountered during simulation, the method `extract_params` uses this function to retrieve the parameters needed for that gate. This approach ensures consistent parameter handling for all types of gates, with no extra logic needed for any parameter type.

A total of 27 commonly used quantum gates (see Table 4) are implemented, covering almost all gates required in practical circuits.

Symbol	Name
X	X (Pauli-X)
Y	Y (Pauli-Y)
Z	Z (Pauli-Z)
H	Hadamard
T	T gate ($\pi/8$ phase gate)
S	S (Phase)
Sdg	Sdg (S-dagger)
SX	SX (sqrt-X)
SXdg	SXdg (sqrt-X-dagger)
$\mathrm{R_X}(\theta)$	X rotation gate
$\mathrm{R_Y}(\theta)$	Y rotation gate
$\mathrm{R_Z}(\theta)$	Z rotation gate
SU(2)	arbitrary single-qubit unitary

(a) Single-qubit gates

Symbol	Name	# Qubits
CX	CNOT	2
CY	Controlled-Y	2
CZ	Controlled-Z	2
CH	Controlled-H	2
SWAP	SWAP	2
iSWAP	iSWAP	2
SU(4)	arbitrary two-qubit unitary	2
CRX(θ)	Controlled-RX	2
CRY(θ)	Controlled-RY	2
CRZ(θ)	Controlled-RZ	2
$\mathrm{R_{XX}}(\theta)$	XX-interaction rotation	2
$\mathrm{R_{YY}}(\theta)$	YY-interaction rotation	2
$\mathrm{R_{ZZ}}(\theta)$	ZZ-interaction rotation	2
CCX	Toffoli (CCX)	3

(b) Multi-qubit gates

Table 4: Quantum gates implemented in `gates.py`. (a) single-qubit set; (b) multi-qubit set.

Each registered gate G receives a `PauliTerm` and gate-specific arguments as input (e.g., qubit indices or angles), and implements the Pauli propagation rule

$$G^\dagger P G = \sum_{Q\in\{I,X,Y,Z\}^{\otimes n}} b_Q Q, \qquad b_Q = \frac{1}{2^n}\mathrm{Tr}[G^\dagger P G Q], \tag{6.1}$$

where G is the gate unitary, P is an input Pauli string (represented as a `PauliTerm`), and $\sum_Q b_Q Q$ is an n-qubit Pauli series (represented as a list of `PauliTerm` objects).

All gate logic relies on core bitwise operations applied to the `key` field of `PauliTerm`, as summarized in Table 5.

Operation name	Python symbol	Math symbol	Example (Python)
bitwise AND	`&`	$a_i \wedge b_i$	`0b1010 & 0b1100 = 0b1000`
bitwise OR	`|`	$a_i \vee b_i$	`0b1010 | 0b1100 = 0b1110`
bitwise XOR	`^`	$a_i \oplus b_i$	`0b1010 ^ 0b1100 = 0b0110`
left shift	`<<`	$a \times 2^k$	`0b0001 << 2 = 0b0100`
right shift	`>>`	$\lfloor a/2^k \rfloor$	`0b1000 >> 2 = 0b0010`

Table 5: Core bitwise operations used in gate implementations. Here, a and b denote arbitrary integers, a_i and b_i denote the i-th bit of a and b respectively, and k is a non-negative integer. For illustration, `0b1010` $= 10$, `0b1100` $= 12$, and `0b1000` $= 8$ in decimal notation.

To illustrate with a concrete example of bitwise Pauli propagation, consider a single-qubit circuit ($n = 1$) with observable $O = X$ and a single T gate applied. The mathematical conjugation is

$$T^\dagger X T = \frac{1}{\sqrt{2}}(X - Y). \tag{6.2}$$

The following code cell shows the implementation of the T gate and illustrates the general pattern used for all gate propagation rules in `gates.py`. Each rule applies bitwise operations directly to the `PauliTerm` representation. For the example input `PauliTerm(1.0, 1, 1)` (which represents the observable X for $n = 1$), each step is annotated with comments to show the explicit calculation.

```python
@QuantumGate.register_gate("t")
def t_gate(pauli_term: PauliTerm, q: int) -> List[PauliTerm]:
    coeff = pauli_term.coeff      # 1.0
    key   = pauli_term.key        # 1, for X (0b01)
    n     = pauli_term.n          # 1

    # Extract x and z bits for target qubit (q = 0)
    x = (key >> q)      & 1       # (1 >> 0) & 1 = 1 (X present)
    z = (key >> (n+q)) & 1        # (1 >> 1) & 1 = 0 (Z absent)

    # For P = I or Z,  T^dagger P T = P
    if (z and not x) or (not x and not z):
        return [pauli_term]

    # Flip z bit to convert X into Y
    key2 = key ^ (1 << (n+q))     # 1 ^ (1 << 1) = 0b11 = 3 (Y)
    c1 = coeff / np.sqrt(2)       # 1 / sqrt(2)
    c2 = -c1                      # z=0, so negative sign

        # Returns: [PauliTerm(1/sqrt(2), 1, 1), PauliTerm(-1/sqrt(2), 3, 1)]
        # which encodes (1/sqrt(2)) X - (1/sqrt(2)) Y
    return [PauliTerm(c1, key, n), PauliTerm(c2, key2, n)]
```

This step-by-step execution confirms that the bitwise operations yield the correct analytic result (Equation (6.2)).

6.3 `propagator.py`: Pauli Back-Propagation

The `propagator.py` module implements the core logic for Pauli back-propagation of observables through quantum circuits. The central class, `PauliPropagator`, initializes from a `Qiskit` circuit, records qubit ordering, and constructs the reverse-ordered gate list for Heisenberg evolution.

The `PauliPropagator.propagate` method implements the algorithm described in Section 4.3. Given an initial Pauli observable (as a single `PauliTerm`), the method iteratively applies each gate in the circuit in reverse using the corresponding Pauli propagation rule from `gates.py`. At each step:

1. Applies the gate to each input `PauliTerm`, expanding it into a list of `PauliTerm` objects (the full Pauli series),
2. Optionally discards terms exceeding the prescribed truncation weight k,
3. Combines coefficients of duplicate Pauli strings,
4. Filters terms with negligible coefficients (below a user-defined tolerance).

The propagation history is recorded gate-by-gate as lists of `PauliTerm`. After full propagation, the final list represents the truncated (k) Pauli expansion of the initial observable, as in Equation (4.5). Propagation can be parallelized via `ProcessPoolExecutor`, with the list of terms partitioned into balanced chunks, so that different processor cores apply the gate in parallel to comparable numbers of `PauliTerm` objects.

The expectation value on the initial state $|\psi_{\text{init}}\rangle$ is then computed via

$$\tilde{f}_U^{(k)}(O) = \sum_{P:|P|\le k} \frac{1}{2^n}\mathrm{Tr}[U_1^\dagger O_1^{(k)} U_1\, P]\, \langle\psi_{\text{init}}|P|\psi_{\text{init}}\rangle, \tag{6.3}$$

where the list of `PauliTerms` and the initial state label are provided as inputs to `expectation_pauli_sum`, which efficiently evaluates this sum using precomputed lookup tables, without constructing full $2^n \times 2^n$ matrices.

The following minimal example demonstrates the usage of `propagate` and `expectation_pauli_sum`. A single-qubit circuit is constructed in `Qiskit` with a T gate; the observable is X, and the initial state is $|+\rangle$. Inline comments indicate the step-by-step computation.

```python
from qiskit import QuantumCircuit
from qiskit.quantum_info import Pauli

# Construct a single-qubit circuit with one T gate
qc = QuantumCircuit(1)
qc.t(0)

n = qc.num_qubits
pauli_label = 'X' # Initial observable: X on qubit 0
key = encode_pauli(Pauli(pauli_label))
init_term = PauliTerm(1.0, key, n)
product_label = "+" * n  # Initial state |+>

# Create the propagator and propagate
prop = PauliPropagator(qc)
# For a single-qubit circuit, max_weight=3 has no effect (shown for completeness)
history = prop.propagate(init_term, max_weight=3)  # [[+1*X], [+0.707*X, -0.707*Y]]

# 0.707*<+|X|+> - 0.707*<+|Y|+>
expectation = prop.expectation_pauli_sum(history[-1], product_label)
print(expectation)  # 0.707,
```

6.4 `monte_carlo.py`: Pauli Path Monte Carlo Sampling

This module implements the algorithm described in Section 5.1, using Monte Carlo sampling of Pauli paths to estimate the MSE of the expectation after truncated Pauli propagation.

Parallel processing is used to maximize throughput: sampling tasks are distributed across CPU cores using `ProcessPoolExecutor`, with the number of concurrent workers set by the user; each worker draws a Pauli path independently.

Sampling results are stored internally and can be saved to or loaded from disk, enabling incremental accumulation of Monte Carlo datasets across multiple runs. The main user-facing interface is the `monte_carlo_samples` method, which:

- Accepts an initial observable O (`PauliTerm`), the number of paths N to sample, and an optional file path for saving samples.
- Loads existing samples if available; generates new samples only as needed.
- Parallelizes sampling jobs and aggregates results.
- Saves the complete dataset (final Pauli terms and per-path weight-exceed flags, weights) for further analysis.

For each path, once any Pauli term exceeds a given threshold k, a persistent flag is set to True. This allows identification of all Pauli paths containing a Pauli string above any threshold k, so the MSE for all k can be estimated from a single batch of samples without resampling for each k.

The method `estimate_mse_for_truncation` computes Monte Carlo estimates of the truncation MSE for each threshold k, as defined in Equation (5.13). It returns $\widehat{\mathrm{MSE}}^{(k)}$ from all paths containing a Pauli string whose maximum weight exceeds k. The overlap d_γ^2 in Equation (5.13) is evaluated as the expectation value of the final propagated Pauli term $\Phi(U)\, s_0$ on the initial state $|\psi_{\mathrm{init}}\rangle$, using `expectation_pauli_sum`. The sample MSE variance $\widehat{\mathrm{Var}}[\widehat{\mathrm{MSE}}^{(k)}]$ for each k is also estimated via Equation (5.22) within this method. Taking the square root yields the standard deviation (Std).

The following example demonstrates how to perform Monte Carlo Pauli-path sampling and estimate the truncation MSE for all weight thresholds k in a generic two-body circuit. The circuit contains 4 sequential repetitions (i.e., each repetition refers to applying all gates in the specified structure once); in each repetition, a randomly chosen two-body rotation $R_{XX}(\theta)$, $R_{YY}(\theta)$, or $R_{ZZ}(\theta)$ with random angle θ is applied successively to the pairs of qubits with indices $(0, 1)$, $(1, 2)$, $(2, 3)$, and $(3, 4)$. (*Assume all necessary classes and functions are imported as in previous code examples.*)

```python
rng      = np.random.default_rng(seed=42)  # Set random seed

# Construct a 5-qubit circuit with 4 repetitions  of random two-body rotations
nq = 5
qc = QuantumCircuit(nq)
pairs = [(0, 1), (1, 2), (2, 3), (3, 4)]
for _ in range(4):
    for q1, q2 in pairs:
        gname  = rng.choice(["rxx", "ryy", "rzz"])
        theta  = rng.uniform(pi/6, 5*pi/6)
        getattr(qc, gname)(theta, q1, q2)

# Define initial observable ZIIII (Z on qubit 4)
init_key  = encode_pauli(Pauli("Z" + "I" * (nq - 1)))
init_term = PauliTerm(1.0, init_key, nq)

prop = PauliPropagator(qc)
mc   = MonteCarlo(qc)

out_dir     = f"results/example"
os.makedirs(out_dir, exist_ok=True)
sample_file = os.path.join(out_dir, "mc_samples.pkl")

# Perform (or load) persistent Monte Carlo sampling for the given observable
mc.monte_carlo_samples(init_term = init_term,
                       M = 100000,
                       sample_file = sample_file,
                       load_existing = True)

# Estimate truncation MSE for all weights k on the initial |00000> state
mse_mc_results_dict = mc.estimate_mse_for_truncation(propagator = prop,
                                                     product_label =  "0" * nq)
```

The circuit generated by the above code is shown in Figure 4 below.

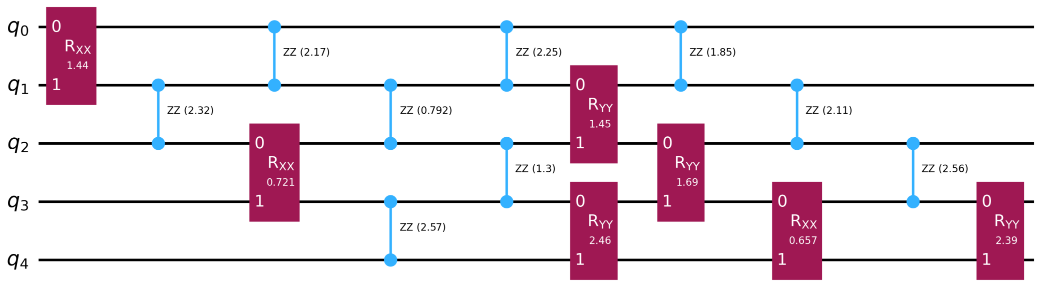


Figure 4: Example 5-qubit, 4-repetitions circuit generated for the Monte Carlo sampling demonstration. R_{XX} and R_{YY} gates are shown as maroon rectangles; each R_{ZZ} gate is displayed as a pair of blue dots connected by a vertical line (`Qiskit` default convention). All gates are labeled by their rotation angle θ (in radians).

The output `mse_mc_results_dict` from the code cell above gives the estimated truncation $\widehat{\mathrm{MSE}}^{(k)}$ and, after taking the square root of the variance, the standard deviation $\widehat{\mathrm{Std}}\left[\widehat{\mathrm{MSE}}_U^{(k)}\right]$ for each weight threshold k:

k	1	2	3	4	5
$\widehat{\mathrm{MSE}}^{(k)}$	0.10396	0.01226	0.00022	0	0
$\widehat{\mathrm{Std}}\left[\widehat{\mathrm{MSE}}_U^{(k)}\right]$	0.00097	0.00035	4.69×10^{-5}	0	0

Table 6: Estimated truncation MSE and standard deviation for each weight cutoff k from Monte Carlo sampling.

6.5 `unit_test/`: Gate-Level Validation

A comprehensive `pytest` suite under `unit_test/` validates every Pauli-propagation gate kernel implemented in `gates.py`. Each test file targets a specific gate family, enabling failures to be traced unambiguously to a propagation rule. All test modules are listed in Table 7.

Test module	**Gate family validated**	**# Test cases**
`test_x_y_z.py`	X, Y, Z	82
`test_h.py`	H	34
`test_t.py`	T	34
`test_rx_ry_rz.py`	R_X, R_Y, R_Z	82
`test_su2.py`	arbitrary single-qubit $SU(2)$ unitaries	70
`test_crx_cry_crz.py`	CRX, CRY, CRZ	166
`test_rxx_ryy_rzz.py`	R_{XX}, R_{YY}, R_{ZZ}	166
`test_cx_cy_cz_ch.py`	CX, CY, CZ, CH	218
`test_swap_iswap.py`	SWAP, iSWAP	114
`test_su4.py`	arbitrary two-qubit $SU(4)$ unitaries	60
`test_ccx.py`	CCX (Toffoli)	434
Total		**1,460**

Table 7: Mapping of unit-test modules to the quantum gates they verify. The final column lists the number of test cases per file. The entire suite completes in approximately 4.5 s on a laptop.

To ensure all gates are correctly implemented, each test file follows a unified structure and repeats each of the following procedures multiple times:

1. **Minimal-support tests:** Performed in the gate's minimal support subspace (i.e., the Hilbert subspace on which the gate acts non-trivially), as an initial validation. For an n-qubit gate ($n = 1, 2, 3$), the relevant `QuantumGate` method is applied to a `PauliTerm` P (randomly sampled from $\{I, X, Y, Z\}^{\otimes n}$), yielding a Pauli series $G^\dagger PG = \sum_Q b_Q Q$ (list of `PauliTerm`). Each $b_Q Q$ is converted to its matrix form and summed to a $2^n \times 2^n$ matrix, then compared to direct conjugation by the gate's full matrix G acting on Pauli matrix for P (i.e., $G^\dagger PG$), ensuring correctness of the core bit-wise operations. All such tests follow the naming convention `test_gatename_rule` (e.g., `test_x_y_z_rule`).
2. **Embedded-qubit tests:** A 6-qubit `PauliTerm` P (randomly sampled from $\{I, X, Y, Z\}^{\otimes 6}$) is selected, together with random target qubit(s) appropriate to the gate type. The relevant `QuantumGate` kernel is applied to the target qubit(s), yielding a Pauli series $\sum_Q b_Q Q$ (list of `PauliTerm`), which is converted and summed to a $2^6 \times 2^6$ matrix. This is compared to the reference conjugation via `Qiskit`, where the embedded gate produces $G^\dagger PG$ (also a $2^6 \times 2^6$ matrix). This procedure detects qubit-indexing errors. All such tests are named `test_gatename_random_embedded` (e.g., `test_x_y_z_random_embedded`).
3. **Random-circuit expectation tests:** For each gate family, a random circuit on n = 3–5 qubits is generated with 3–7 randomly placed gates from that family. Both the observable O (`PauliTerm`, sampled from $\{I, X, Y, Z\}^{\otimes n}$) and the initial state $|\psi_{\text{init}}\rangle$ (sampled from $\{0, 1, +, -, r, l\}^{\otimes n}$) are used. The expectation value is computed using the `PauliPropagator` class (without truncation; see Section 6.3 for usage), and compared against the result from `Qiskit`'s statevector expectation method. This provides an end-to-end check covering kernel logic, gate embedding, propagation, and expectation calculation. All such tests are named `test_gatename_random_circuits` (e.g., `test_x_y_z_random_circuits`).

6.6 `integration_test/`: End-to-End Circuit Validation

The `integration_test` directory contains a suite of end-to-end tests designed to validate the correctness and robustness of the package on complex quantum circuits. All tests compare the expectation values computed by `PauliPropagator` class (without truncation) against exact statevector simulation (`Qiskit`). Specifically, the integration test suite includes the following modules:

- `test_propagator_random.py`: For each of 200 independent trials, all circuit parameters—including the number of qubits, gate types, gate parameters, observable, and initial state—are randomly sampled to expose any potential implementation bugs arising from edge cases or developer habits.
- `test_staircase_random_su4.py`: Focuses on two-dimensional staircase random $SU(4)$ circuits (structure detailed in Section 7) on various grid sizes (n_x, n_y), using initial state $|+\rangle^{\otimes n}$ and observable X_0.
- `test_su4_kak_consistent.py`: Validates the consistency of expectation values between original random $SU(4)$ staircase circuits and their Cartan (KAK) decomposed forms (with KAK decomposition described in detail in Section 7), for a variety of grid sizes, random observables, and random initial states.
- `test_kicked_ising.py`: Tests are conducted on Trotterized transverse-field Ising (TFI) circuits with the connectivity topology of the IBM Eagle chip (details of the TFI model and topology are provided in Section 8), using various circuit repetitions and subcircuits of different sizes (i.e., rather than the full 127-qubit device) to permit exact expectation value calculation. The observable, initial state, and circuit parameters are all randomized.

Because `PauliPropagator` matches statevector results exactly in all small-scale circuits when run without truncation, the correctness of the implementation is fully established. Theoretical error bounds (Theorem 4.1) then guarantee that, when truncation is required for larger circuits, the resulting expectation estimates remain controlled and reliable. These validations ensure that the `pauli_pkg` package can be confidently applied in the analysis of quantum circuits presented in the following sections. Together, the `unit_test` and `integration_test` suites provide robust coverage for all gate propagation rules and proves full compatibility of `pauli_pkg` with `Qiskit`.

A time benchmark and truncation weight benchmark on a 4×4 staircase random SU(4) circuit (topology as in Fig. 5, shown there for 6×6) are additionally performed, with results shown in Appendix A.

7 Staircase Random $SU(4)$ Circuit

To benchmark the power and limitations of the Pauli propagation algorithm, this section focuses on circuits arranged in a 6×6 two-dimensional staircase topology (see Fig. 5, implemented in `circuit_topologies.py`) with many repetitions (i.e., repeatedly applying the staircase pattern to the qubits), where each pair of neighboring qubits is connected by a random $SU(4)$ gate. The choice of circuit is motivated by:

- The topology and gate type ensure that a Pauli observable supported on a single qubit rapidly spreads its support to all qubits after only one repetition, destroying locality and creating strong entanglement. Circuits with 36 qubits already exceed the capabilities of exact statevector simulation, and Zhang *et al.* [7] have shown that such circuits cannot be simulated by tensor network methods due to the rapid growth of entanglement.
- Each gate in this circuit is independently and uniformly sampled from $SU(4)$, which satisfies Definition 2. Therefore, the entire circuit is drawn from a locally scrambling circuit ensemble.

7.1 Circuit Ensemble and Haar Sampling

In practice, each gate is sampled from the Haar measure [8] on $SU(4)$ using the following construction. Let $Z \in \mathbb{C}^{4\times 4}$ be a random matrix whose entries are independent and identically distributed standard complex Gaussian random variables. Perform the unique QR decomposition

$$Z = QR, \tag{7.1}$$

where $Q \in U(4)$ and R is upper triangular with positive real diagonal entries. Define

$$D = \operatorname{diag}\left(\frac{R_{11}}{|R_{11}|}, \frac{R_{22}}{|R_{22}|}, \frac{R_{33}}{|R_{33}|}, \frac{R_{44}}{|R_{44}|}\right), \tag{7.2}$$

and set

$$U = QD^{-1} \det(QD^{-1})^{-\frac{1}{4}}. \tag{7.3}$$

This guarantees $U \in SU(4)$ and that U is Haar distributed.

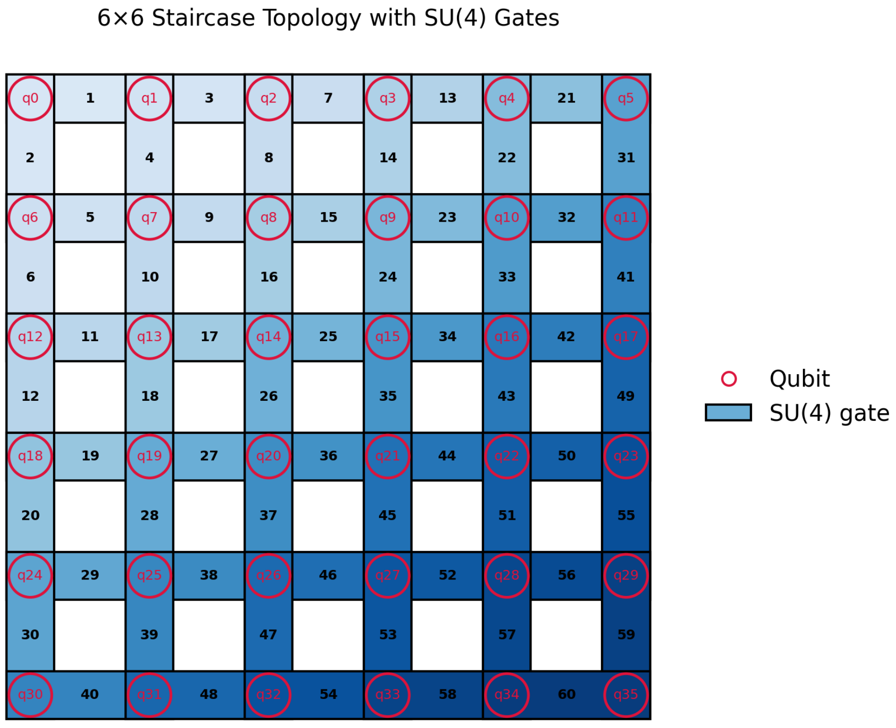


Figure 5: Schematic of the 6×6 staircase topology for a single repetition (contains 60 gates). Each blue rectangle represents a random $SU(4)$ quantum gate, with the number inside indicating the placement order: lighter blue corresponds to earlier gates, darker blue to later ones. Crimson circles denote qubits, with their indices shown inside each circle. Qubit 0 is at the upper left corner and qubit 35 at the lower corner.

7.2 KAK Decomposition and Elementary Gate Sequence

Each $SU(4)$ gate is further decomposed via the KAK decomposition into single-qubit rotations and standard two-qubit interactions (R_{XX}, R_{YY}, R_{ZZ}). For any $U \in SU(4)$, the KAK decomposition takes the mathematical form [9]:

$$U = (K_0^F \otimes K_1^F)\, R_{XX}(\alpha_x) R_{YY}(\alpha_y) R_{ZZ}(\alpha_z)\, (K_0^B \otimes K_1^B) \tag{7.4}$$

Here, α_x, α_y, and α_z are real angle parameters in the Weyl chamber (i.e., $\frac{\pi}{4} \geq \alpha_x \geq \alpha_y \geq |\alpha_z|$). Each K_j^D ($j = 0, 1$; $D = F, B$) is a single-qubit $SU(2)$ rotation acting on qubit j from the front (F) or back (B) of $R_{XX}R_{YY}R_{ZZ}$, and can be further decomposed as [10]

$$K_j^D = R_z(\lambda_j^D)\, R_y(\theta_j^D)\, R_z(\phi_j^D) \tag{7.5}$$

with real Euler angles λ_j^D, θ_j^D, and ϕ_j^D.

The KAK decomposition process is illustrated in Figure 6.

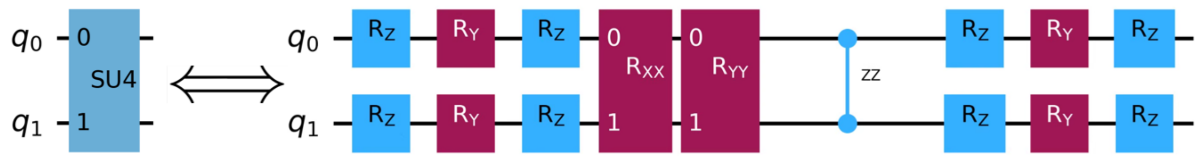


Figure 6: KAK decomposition: a single $SU(4)$ gate is exactly represented by 12 single-qubit Euler rotations and three two-qubit interactions (R_{XX}, R_{YY}, R_{ZZ}).

7.3 Simulation Parameters, Truncation and Results

All subsequent experimental results are obtained on staircase random $SU(4)$ circuits, where each $SU(4)$ gate (see Figure 5) is replaced by a sequence of 15 elementary gates from KAK decomposition(see Figure 6).

Note that only the full 15-gate sequence corresponds to a sample from a locally scrambling distribution; individual gates in the KAK sequence do not. Therefore, applying truncation after each of the 15 gates can only decrease the accuracy of estimation and breaks the assumptions of Theorem 4.1, so the resulting error bound is no longer guaranteed to hold. Nevertheless, KAK decomposition and gate-by-gate truncation remain beneficial because:

- A generic $SU(4)$ gate can expand a single Pauli string into up to 15 terms in one step. Truncating after each elementary gate continually prunes high-weight terms, which greatly reduces the number of retained Pauli terms, saving both time and memory.
- Generic two-qubit $SU(4)$ gates are not native to hardware. Real devices provide arbitrary single-qubit rotations and a fixed entangling gate (e.g., CX); all circuits are compiled into this universal set. The KAK decomposition rewrites any $SU(4)$ as three entangling gates interleaved with single-qubit rotations, which can be implemented on such hardware, ensuring the simulation circuit mirrors real device.

Experiments use KAK-decomposed staircase random $SU(4)$ circuits with repetitions Rep $= 1, 2, 3, 4, 5$. During Pauli propagation, truncation at weight $k = 1, 2, 3, 4$ is performed after each gate, as in [2]. The initial state is $|\psi_{\text{init}}\rangle = |0\rangle^{\otimes 36}$. The observable is $P = Z_{35} \otimes I_{34} \otimes \cdots \otimes I_0$, after just one circuit repetition, P spreads into a Pauli series supported on all qubits.

Different circuit repetitions and truncation weights yield Pauli propagation runtimes as shown in Fig. 7, increasing exponentially in the truncation weight k, but only polynomially with circuit depth. The time complexity scales roughly as $O(Ln^k)$. Exact timings are listed in Appendix B, Table 9.

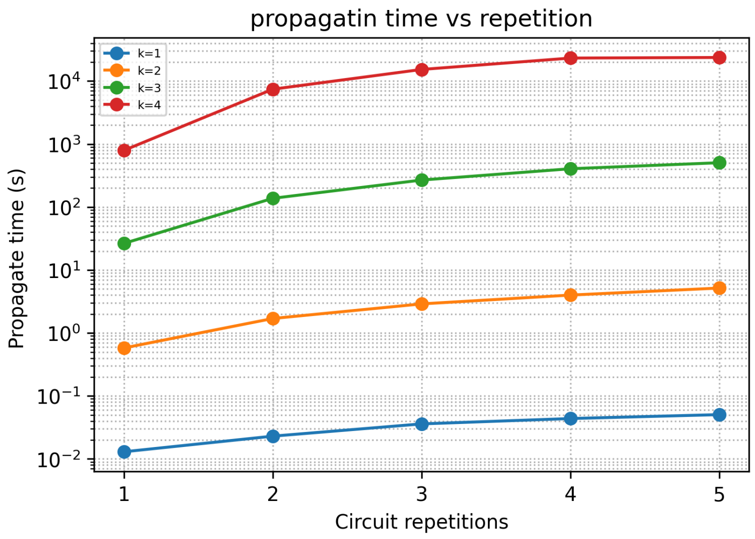


Figure 7: Pauli propagation runtime versus circuit repetition for truncation weights $k = 1, 2, 3, 4$.

Figure. 8 demonstrates that the magnitude of the expectation value decays rapidly with increasing circuit repetitions, dropping from order 10^{-1} after a single layer to around 10^{-2} for $L \geq 3$ across all truncation weights k. Successive random $SU(4)$ layers transform an initially local observable into a Pauli series, whose average Pauli weight increases with repetition (see Fig. 9a). As the observable is delocalized into higher-weight Pauli terms, its expectation magnitude with respect to the initial state is strongly suppressed, consistent with the prediction of Equation (4.14).

Due to the prohibitive computational cost of simulating such circuits (as shown in Table 9, a single $k = 4$ Pauli propagation run with 4 repetitions takes over 23,000 seconds, or approximately 6.5 hours), each value in Figure 8 is obtained from a single random circuit instance per repetition. As a result, statistical fluctuations are possible, but the overall decay and delocalization trends remain representative.

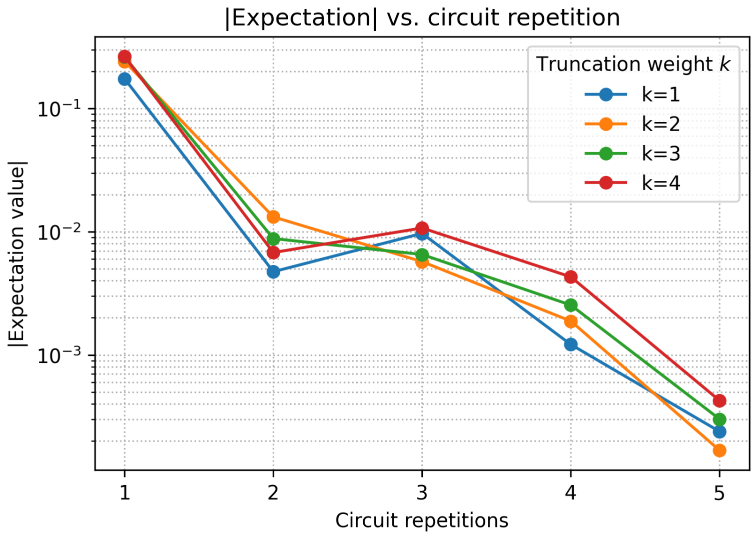


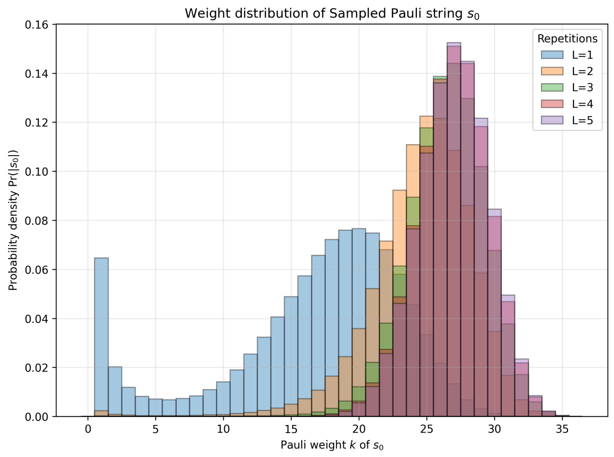
Figure 8: Magnitude of the expectation estimate $|\tilde{f}_U^{(k)}(O)|$ v.s. circuit repetitions for different truncation weights.

Monte Carlo sampling over Pauli paths is performed for this system, with the number of samples for each circuit repetition listed in Table 8.

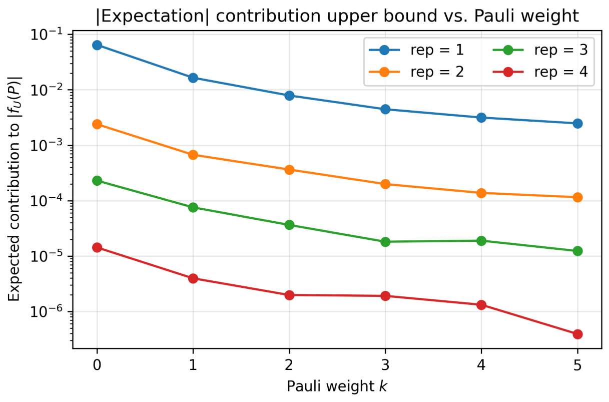
Repetition	1	2	3	4	5
Samples ($\times 10^5$)	5.0	15.0	15.0	37.0	1.0

Table 8: Number of Monte Carlo samples used for each circuit repetition. For $L = 5$, only 10^5 samples were generated, solely for plotting Fig. 9a, and not for estimating MSE.

As shown in Fig. 9a, the empirical weight distribution obtained from Monte Carlo samples shifts toward higher weights and becomes narrower as repetition increases, reflecting rapid delocalization of Pauli strings. The differences among repetitions 1, 2, and 3 are significant, whereas for repetitions 4 and 5 the distributions are closely aligned; beyond a certain number of repetitions, the distribution shape remains nearly unchanged. Fig. 9b shows the product of the probability densities from Fig. 9a and $(2/3)^{k/2}$, illustrating, for each repetition, an upper bound from Eq. (4.14) on how much Pauli strings of weight k can contribute to the magnitude of the expectation value.



(a) Weight distribution of the final Pauli string s_0 from Monte Carlo samples with repetitions 1– 4.



(b) Estimated maximum contribution to the expectation value from Pauli strings of weight k.

Figure 9

As shown in Fig. 10, the MSE decreases rapidly with increasing k. Monte Carlo estimates for this concrete circuit instance substantially outperform the theoretical upper bound, even though truncation is applied after each elementary gate in the $SU(4)$ decomposition, rather than only after each $SU(4)$ gate.

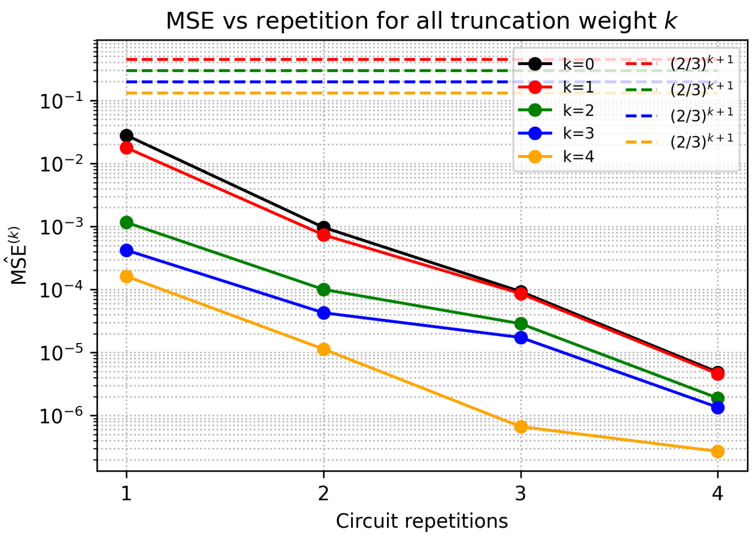


Figure 10: Estimated truncation MSE versus circuit repetitions for various truncation weights k. Here, $k = 0$ corresponds to truncate everything. Dashed lines indicate the analytic upper bound predicted by Theorem 4.1.

Figure 10 only shows up to 4 circuit repetitions because:

- The probability of sampling Pauli paths with a nonzero contribution to the MSE estimate at high weight drops sharply. This is due to the statistical structure of the random SU(4) staircase circuit: as repetition increases, the support of the propagated Pauli string rapidly grows and can be randomized to any of $\{I, X, Y, Z\}$, but the chance that a sampled path γ contains a Pauli term with large weight k and ends with a final Pauli $s_0^{(\gamma)}$ containing only I or Z (required for a nonzero $d_\gamma^2 = \mathrm{Tr}[s_0^{(\gamma)} \, |0\rangle^{\otimes 36} \langle 0|]$ to calculate $\widehat{\mathrm{MSE}}^{(k)}$, see Equation 5.13) decreases exponentially, making it very difficult to obtain effective samples.
- The time required to sample each Pauli path increases substantially with repetition. For example, generating 3.7×10^5 samples at repetition 4 took about 13.5 hours on 8 processors.

Since the number of effective samples for estimating MSE drops exponentially, the required sampling time increases further. As a reference, according to private correspondence with the original author of [2], 100 million samples per data point were used to obtain valid MSE estimates in their 8×8 system. For repetition 5 and beyond, MSE estimation is practically infeasible with this code for the 6×6 system considered here.

Table 10 (Appendix B) summarizes the observable estimate $\tilde{f}_U^{(k)}(O)$, the Monte Carlo mean-squared error $\widehat{\mathrm{MSE}}_U^{(k)}$, and the corresponding standard deviation $\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}]$ for truncation weights $k = 1$–4 and repetitions $L = 1$–4. The following points highlight the main findings:

- Entries for circuits with one repetition satisfy

$$\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}] \lesssim 0.1\, \widehat{\mathrm{MSE}}_U^{(k)}, \tag{7.6}$$

  indicating that the MSE is reliable in this regime. Moreover, for $k = 4$, $\sqrt{\widehat{\mathrm{MSE}}_U^{(4)}}$ is much smaller than the expectation value, confirming the accuracy of the expectation estimate.

- To quantify the effect of truncation, consider the dimension-less relative error ratio:

$$\varepsilon_{\mathrm{rel}} = \frac{\sqrt{\widehat{\mathrm{MSE}}_U^{(k)}}}{|\tilde{f}_U^{(k)}(O)|}. \tag{7.7}$$

  For example, in the shallowest circuit with $k = 4$, this ratio is as small as 5%, showing that low-weight truncation can offer accurate results. However, as the circuit deepens—even with $k = 4$ maintained—the numerator $\sqrt{\widehat{\mathrm{MSE}}_U^{(k)}}$ decreases more slowly than the denominator $|\tilde{f}_U^{(k)}(O)|$, so the ratio inflates.

Moreover, the standard deviation of $\widehat{\mathrm{MSE}}_U^{(k)}$ also grows (e.g., for Rep. 4, $k = 3$, $\widehat{\mathrm{MSE}}_U^{(k)} = 1.351 \times 10^{-6}$ while $\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}] = 6.043 \times 10^{-7}$), indicating that the apparently large $\varepsilon_{\mathrm{rel}}$ reflects low signal-to-noise due to insufficient sampling rather than a failure of the propagation method.

- In the (Rep $= 3, 4$) columns, only one Pauli path of weight > 4 with $d_\gamma^2 \neq 0$ is sampled. With a single effective sample, the associated standard deviation collapses to the same value as the MSE. In such rare-event regimes, a larger sample size is required for a meaningful variance and MSE estimate.

In all cases studied, the empirically estimated MSE for Pauli propagation remains consistently below the theoretical upper bound. More extensive sampling would be required to draw more general conclusions.

8 Kicked Transverse-Field Ising Circuit

The TFI model describes a system of spin-$\frac{1}{2}$ particles (e.g., electron spins) arranged on a two-dimensional lattice [11], with each site hosting one spin under a uniform transverse magnetic field. It is a paradigmatic model in quantum many-body physics. Neighboring spins interact via an Ising coupling that favors alignment along the z-axis, so the Hamiltonian is

$$H_{ZZ} = -J \sum_{\langle i,j \rangle} Z_i Z_j, \tag{8.1}$$

where $J > 0$ is the Ising coupling strength, Z_i is the Pauli Z operator acting on site i, and $\langle i, j \rangle$ denotes all pairs of neighboring sites.

A uniform transverse magnetic field along the x-direction induces quantum fluctuations that flip spins, competing with the Ising order. This contribution to the Hamiltonian is

$$H_X = -h_x \sum_i X_i, \tag{8.2}$$

where h_x is the field strength and X_i is the Pauli X operator acting on site i.

Combining these terms, the full Hamiltonian is

$$H_{\mathrm{TFI}} = H_{ZZ} + H_X. \tag{8.3}$$

The dynamics are governed by the Schrödinger equation,

$$i \frac{d}{dt} |\psi(t)\rangle = H_{\mathrm{TFI}} |\psi(t)\rangle, \tag{8.4}$$

with solution

$$|\psi(t)\rangle = e^{-iH_{\mathrm{TFI}}t} |\psi(0)\rangle = e^{-i(H_{ZZ}+H_X)t} |\psi(0)\rangle. \tag{8.5}$$

The operators H_{ZZ} and H_X do not commute, so the exact evolution $e^{-i(H_{ZZ}+H_X)T}$ cannot be factorised into two separate exponentials. A direct approach would require manipulating a $2^n \times 2^n$ matrix, which is infeasible for large n and cannot be mapped to native quantum gates. To address this, discretize the total time T into L_t steps of duration $\tau = T/L_t$, and apply the first-order Trotter formula [12]:

$$e^{-i(H_{ZZ}+H_X)\tau} \approx e^{-iH_X\tau} e^{-iH_{ZZ}\tau}, \qquad \text{error } \mathcal{O}\big(\tau^2 \|[H_X, H_{ZZ}]\|\big). \tag{8.6}$$

The full evolution is then approximated by

$$U(T) = e^{-i(H_{ZZ}+H_X)T} \approx \Big[e^{-iH_X\tau} e^{-iH_{ZZ}\tau}\Big]^{L_t}. \tag{8.7}$$

Each sub-exponential involves only local terms: $e^{-iH_X\tau}$ is implemented by parallel single-qubit $R_X(2h_x\tau)$ rotations, and $e^{-iH_{ZZ}\tau}$ by parallel two-qubit $R_{ZZ}(2J\tau)$ gates on all nearest-neighbour pairs, where

$$R_X(\theta) = \exp\!\big(-i\tfrac{\theta}{2}X\big) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \quad R_{ZZ}(\theta) = \exp\!\big(-i\tfrac{\theta}{2} Z \otimes Z\big). \tag{8.8}$$

Equivalently, the evolution corresponds to a piecewise-constant (kicked) Hamiltonian,

$$H(t) = \begin{cases} H_X, & t \in [k\tau, (k+\frac{1}{2})\tau), \\ H_{ZZ}, & t \in [(k+\frac{1}{2})\tau, (k+1)\tau), \end{cases} \qquad k = 0, \ldots, L_t - 1, \tag{8.9}$$

which yields a depth-$2L_t$ circuit of alternating R_X and R_{ZZ} layers, efficiently executable both on quantum hardware (as native gates) and in classical simulation (as local state-vector updates, avoiding global matrix manipulation).

8.1 Experimental Protocol and Benchmark

This study considers the implementation of kicked Ising dynamics on IBM's 127-qubit Eagle quantum processor, which arranges qubits on a heavy-hex graph—a planar structure derived from a hexagonal lattice, where each qubit has two or three nearest neighbours (see Fig. 11, implemented in `circuit_topologies.py`).

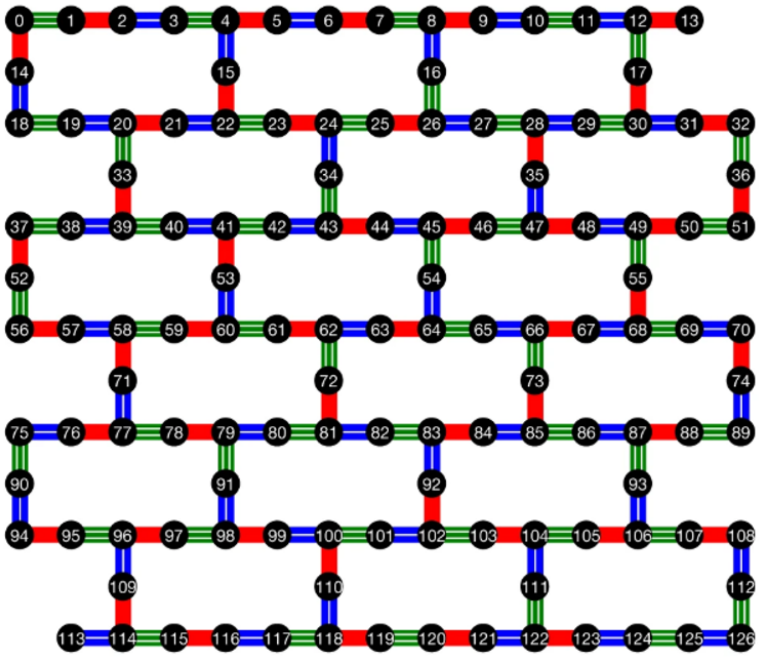


Figure 11: Heavy-hex topology of IBM's 127-qubit Eagle processor. Each node is a qubit, and edges indicate nearest-neighbour connectivity. Adapted from [13]

In one representative experiment on such hardware [13], the initial state is $|0\rangle^{\otimes 127}$ and the observable is Z_{62}, selected because qubit 62 is located near the geometric center of the heavy-hex lattice. After 20 Trotter steps, the evolution entangles qubit 62 with the entire system, so the final state exhibits global correlations; measuring Z_{62} thus becomes challenging for classical simulation methods such as tensor networks.

Each Trotter step in the experimental protocol consists of:

1. Applying a single-qubit $R_X(\theta_h)$ rotation to every qubit, with θ_h varied over 11 values between 0 and $\pi/2$ (corresponding to different magnetic field strengths).

2. Applying $R_{ZZ}(\theta_{ZZ})$ gates to every nearest-neighbour pair, with all $\theta_{ZZ} = -\pi/2$. Hardware forbids simultaneous two-qubit gates that share a qubit, so the edges are partitioned into three colours (red, green, blue in Fig. 11) such that no two edges of the same colour overlap; all R_{ZZ} gates of a given colour are applied in parallel, yielding three consecutive sub-layers per Trotter step. Since the R_{ZZ} gates commute, the order of these sub-layers does not affect the outcome in the noiseless circuit.

This experimental protocol provides a stringent test for classical simulation methods. The Pauli propagation algorithm is therefore benchmarked under the same protocol, enabling direct comparison with error-mitigated experimental data from the Eagle processor, taken here as ground truth. As shown in Fig. 12, Pauli propagation with moderate k (≥ 5) closely tracks the results from the quantum processor, whereas MPS exhibits increasing deviation at larger angles due to entanglement growth.

Remarkably, generating a single MPS expectation value at fixed θ_h was performed on a 64-core, 2.45 GHz processor with 128 GB memory and required 30 hours [13]. In contrast, Pauli propagation at truncation weight $k = 7$ needed only $\sim$20 minutes on an Intel i7-13700H (14 cores, 3.7 GHz, 32 GB; serial execution) for the most challenging angle, $\theta_h = 0.8$ rad ($\approx \pi/4$), where each Trotter step (at $\theta_h = \pi/4$) maximally generates superposition and entanglement. The $k = 7$ estimate agrees with the Eagle experiment at nearly all θ_h values within uncertainty, substantially outperforming the MPS method: Pauli propagation is about **90 times** faster while achieving closer agreement with the quantum processor.

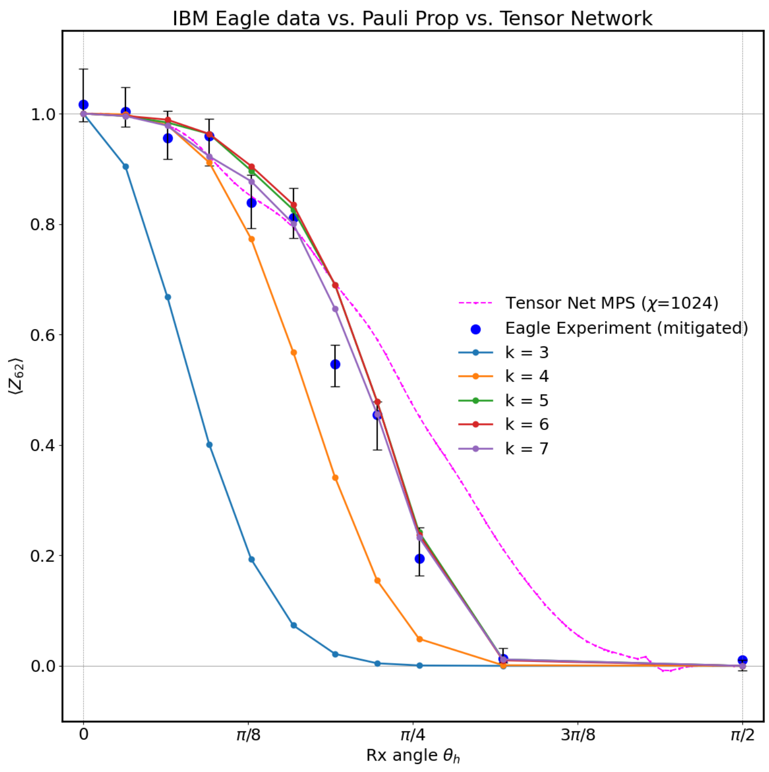


Figure 12: Comparison of Eagle experiment, tensor network, and Pauli propagation estimates of the expectation value $\langle Z_{62} \rangle$ as a function of θ_h after 20 Trotter steps. Eagle experimental results are error-mitigated (blue dots, with error bars), MPS uses bond dimension $\chi = 1024$ (magenta dashed), and Pauli propagation results are shown for truncation weights $k = 3$–7. Eagle experimental and MPS data from [14].

This performance demonstrates that, even for circuits not sampled from a locally scrambling circuit ensemble (here all θ_{ZZ} are identically set to $-\pi/2$), low-weight truncation Pauli propagation can significantly exceed theoretical expectations.

Additional results on runtime versus Trotter steps (circuit repetitions) and the Monte Carlo sampled Pauli string s_0 weight distribution are provided in Appendix C.

9 Further Research Directions

Quantum optimisation: Pauli propagation can quickly estimate cost functions for variational quantum algorithms, especially when cost Hamiltonians are sums of low-weight Pauli terms. This enables fast parameter sweeps in hybrid quantum–classical optimisation, speeding up real-world applications such as Quadratic Unconstrained Binary Optimization in finance.

Noisy-device modelling: By adding simple noise models (e.g., depolarising or amplitude damping channels) between gates, the framework can predict how real quantum hardware will perform. This makes it possible to compare devices, test error-mitigation methods, and guide hardware improvements—all without running time-consuming experiments.

10 Conclusion

This report presents `pauli_pkg`, a Python implementation of the low-weight Pauli-propagation framework that enables classically efficient estimation of observable expectation in deep, high-qubit quantum circuits. Exploiting a compact bit-mask encoding for Pauli operators, a registry-based gate kernel abstraction, the package realises gate–by–gate Heisenberg evolution with on-the-fly weight truncation and certified Monte Carlo error quantification. Comprehensive unit and integration tests confirm exact agreement with state-vector simulation, while large-scale benchmarks demonstrate the algorithm's superior scaling: on a 6×6 random $SU(4)$ staircase circuit, truncation at weight $k = 4$ enables efficient and accurate observable estimation in low-repetition circuits— beyond the reach of state-of-the-art tensor network techniques [7]. For high-repetition circuits, however, the reliability of both the expectation estimates and Monte Carlo error bars may be limited by insufficient sampling, and further investigation is needed to assess accuracy in this regime. For IBM's 127-qubit kicked-Ising experiment, Pauli propagation with $k \leq 7$ reproduces mitigated quantum hardware data to within experimental error yet is two orders of magnitude faster than state-of-the-art MPS simulations. These results empirically validate the theoretical MSE bound, reveal rapid exponential suppression of high-weight Pauli contributions, and substantiate low-weight truncation as a powerful heuristic even for non–scrambling circuits. These properties make Pauli propagation a practical tool for simulating and benchmarking large quantum circuits on today's noisy quantum computers.

References

[1] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders, *Efficient quantum algorithms for simulating sparse Hamiltonians*, Communications in Mathematical Physics **270**, 359–371 (2007).

[2] 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).

[3] U. Schollwöck, *The density-matrix renormalization group in the age of matrix product states*, Annals of Physics **326**, 96–192 (2011).

[4] D. Grier and L. Schaeffer, *The classification of Clifford gates over qubits*, Quantum **6**, 734 (2022), arXiv:1603.03999.

[5] J. L. W. V. Jensen, *Sur les fonctions convexes et les inégalités entre les valeurs moyennes*, Acta Mathematica **30**, 175–193 (1906).

[6] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, *Quantum computing with Qiskit*, arXiv:2405.08810 (2024).

[7] 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).

[8] F. Mezzadri, *How to generate random matrices from the classical compact groups*, Notices of the AMS **54**, 592–604 (2007).

[9] N. Khaneja, R. Brockett, and S. J. Glaser, *Time optimal control in spin systems*, Phys. Rev. A **63**, 032308 (2001).

[10] J. J. Sakurai and J. Napolitano, *Modern Quantum Mechanics*, 3rd Edition, Cambridge University Press, Section 3.3.3, p. 166 (2020).

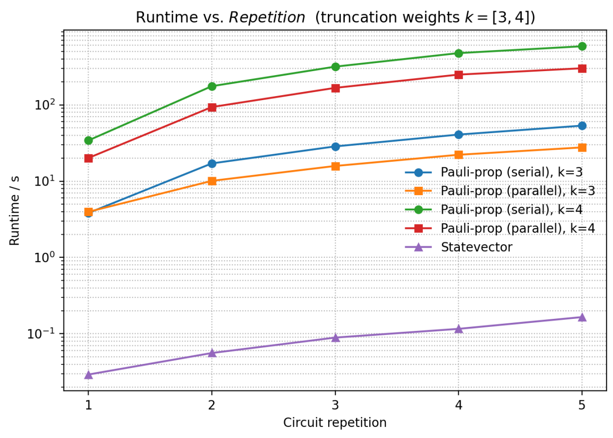
[11] S. Suzuki, J.-i. Inoue, and B. K. Chakrabarti, *Quantum Ising Phases and Transitions in Transverse Ising Models*, Springer (2013).

[12] H. F. Trotter, *On the product of semi-groups of operators*, Proc. Amer. Math. Soc. **10**, 545–551 (1959).

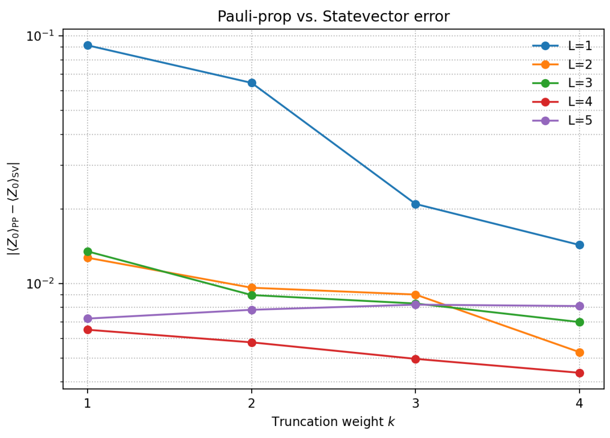
[13] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. van den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, and A. Kandala, *Evidence for the utility of quantum computing before fault tolerance*, Nature **618**, 500–505 (2023).

[14] 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.

A Appendix: 4×4 $SU(4)$ Benchmark Results



(a) Runtime versus circuit repetition L.



(b) Runtime versus truncation weight k for different repetitions.

Figure 13: Pauli propagation and statevector simulation benchmarks on a 4×4 staircase random SU(4) circuit (Intel i7-13700H, 32 GB memory). Observable Z_{15}, initial state $|0\rangle^{\otimes 16}$. Left: runtime vs. repetition for $k = 3, 4$. Right: error vs. truncation weight k.

B Appendix: 6×6 $SU(4)$ Circuit Detailed Numerical Results

k	Rep. 1	Rep. 2	Rep. 3	Rep. 4	Rep. 5
1	0.0130	0.0230	0.0360	0.0438	0.0506
2	0.5788	1.6985	2.8973	3.9998	5.1716
3	26.472	137.56	268.24	405.58	505.35
4	794.23	7432.2	15299	23190	23695

Table 9: Propagation time (seconds) for different truncation weights k at circuit repetitions Rep. 1–5.

k	Rep. 1			Rep. 2		
	$\tilde{f}_U^{(k)}(O)$	$\widehat{\mathrm{MSE}}_U^{(k)}$	$\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}]$	$\tilde{f}_U^{(k)}(O)$	$\widehat{\mathrm{MSE}}_U^{(k)}$	$\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}]$
1	-0.1750	1.773×10^{-2}	1.866×10^{-4}	-0.00474	7.340×10^{-4}	2.211×10^{-5}
2	-0.2424	1.158×10^{-3}	4.810×10^{-5}	0.01320	1.000×10^{-4}	8.165×10^{-6}
3	-0.2632	4.200×10^{-4}	2.898×10^{-5}	0.00880	4.267×10^{-5}	5.333×10^{-6}
4	-0.2654	1.620×10^{-4}	1.800×10^{-5}	0.00678	1.133×10^{-5}	2.749×10^{-6}

(a) Repetitions 1 and 2

k	Rep. 3			Rep. 4			Rep. 5
	$\tilde{f}_U^{(k)}(O)$	$\widehat{\mathrm{MSE}}_U^{(k)}$	$\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}]$	$\tilde{f}_U^{(k)}(O)$	$\widehat{\mathrm{MSE}}_U^{(k)}$	$\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}]$	$\tilde{f}_U^{(k)}(O)$
1	-0.00969	8.600×10^{-5}	7.572×10^{-6}	-0.00121	4.595×10^{-6}	1.114×10^{-6}	-0.00024
2	-0.00572	2.867×10^{-5}	4.372×10^{-6}	-0.00188	1.892×10^{-6}	7.151×10^{-7}	0.00017
3	-0.00653	1.733×10^{-5}	3.399×10^{-6}	-0.00254	1.351×10^{-6}	6.043×10^{-7}	0.00030
4	-0.01072	6.667×10^{-7}	6.667×10^{-7}	-0.00429	2.703×10^{-7}	2.703×10^{-7}	0.00043

(b) Repetitions 3, 4, and 5

Table 10: Expectation $\tilde{f}_U^{(k)}(O)$, mean squared error $\widehat{\mathrm{MSE}}_U^{(k)}$, and estimated standard deviation $\widehat{\mathrm{Std}}[\widehat{\mathrm{MSE}}_U^{(k)}]$ for truncation weights $k = 1$–4 at repetitions 1–4. For repetition 5, only the observable estimate is shown; MSE and standard deviation estimation are practically infeasible. The initial observable is $O = Z_{35} \otimes I_{34} \otimes \cdots \otimes I_0$, and the initial state is $|\psi_{\text{init}}\rangle = |0\rangle^{\otimes 36}$.

C Appendix: Transverse-Field Ising Circuit Additional Results

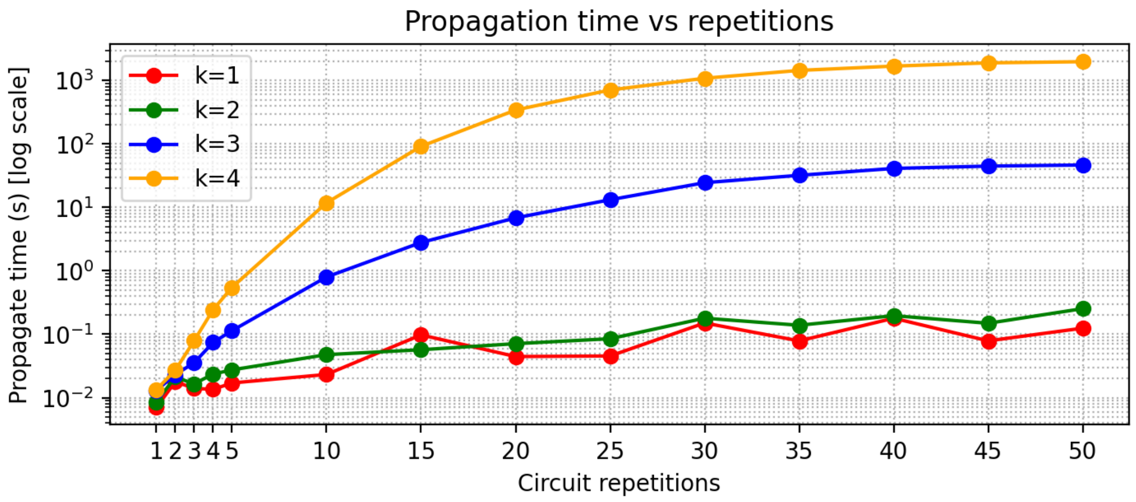


Figure 14: Run time for propagating the observable Z_{62} on a TFI circuit as a function of circuit repetitions, with $R_X(\theta_X)$ rotation angle set to $\pi/2$ and $R_{ZZ}(\theta_{ZZ})$ angle set to $\pi/4$.

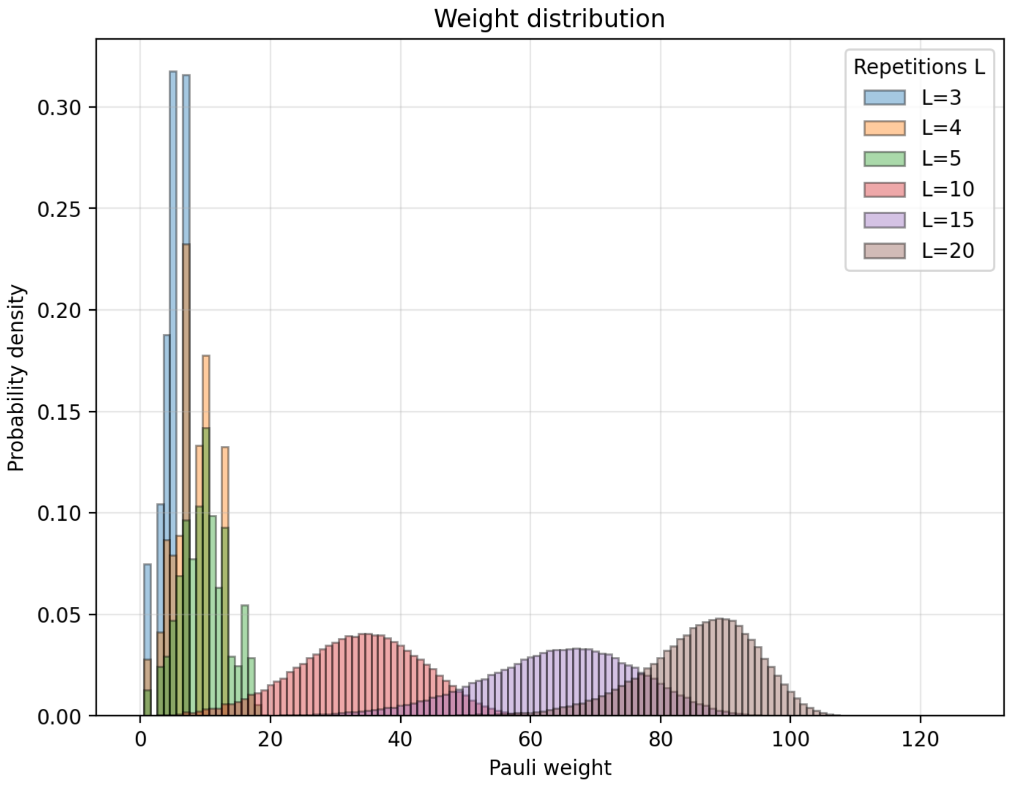


Figure 15: Distribution of the terminating Pauli string s_0 weights obtained via Monte Carlo sampling for TFI circuits with repetitions $L = 3, 4, 5, 10, 15, 20$. Each circuit sets the $R_X(\theta_X)$ rotation angle to $\pi/3$ and the $R_{ZZ}(\theta_{ZZ})$ angle to $-\pi/2$. For each repetition, 2×10^5 samples were collected.

D Appendix: Declaration of AI Usage

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