**Slide 1: Title**

"Hello my name is Steven Ma. Today, I’ll be presenting my project work on classical simulation of quantum circuits using Pauli propagation. ~~This method focuses on low-weight Heisenberg evolution and provides certified Monte Carlo error bounds. I was supervised by Professor Hamza Fawzi, and this work is part of my MPhil in Data Intensive Science at Cambridge, completed in July 2025. My talk will cover the motivation, theoretical background, implementation details, benchmarking, and experimental results on state-of-the-art quantum hardware."~~

**2 (Why Quantum Simulation?):**

So, why do we care about quantum simulation? The answer is that predicting the properties of molecules—like finding new catalysts or drugs—requires us to simulate quantum systems. But here’s the challenge: simulating many-body quantum systems on a classical computer is exponentially hard. For n qubits, the Hilbert space dimension is 2^n, which means that brute-force classical methods can only handle systems with up to about 35 qubits. This limitation motivated Feynman to propose that if we really want to simulate nature, we need quantum computers. My project is about pushing the boundary of what’s possible classically, before we reach that quantum advantage.

**3 (Quantum Advantage: A Moving Target):**

Only when a quantum computer, running a quantum algorithm, completes a given task with less time and resources than any known classical algorithm—including all classical simulations of quantum algorithms—do we say that the quantum computer has achieved quantum advantage for this task.

But this is a moving target, because every time classical algorithms improve, the bar for quantum advantage gets higher. New classical simulation methods—like Pauli propagation, tensor networks,—keep raising that bar. So, for quantum computers to be practically valuable, they have to outperform the very best classical techniques we can come up with. That’s why developing strong classical simulators isn’t just an academic exercise; it’s essential for benchmarking quantum hardware and truly understanding when, and for which problems, quantum devices have a real advantage.

**4（Statevector Simulation: Brute-Force Approach）**

Statevector simulation is the most direct way to simulate a quantum system: we just store the full quantum state as a huge vector of 2^n complex numbers, where n is the number of qubits. Every gate acts as a massive matrix on this statevector. The upside is that this method is exact and works for any quantum circuit. But the downside is exponential memory. Even with today’s supercomputers, we can only push this method to about 35 qubits.

**5 Tensor Network Methods: Exploiting Structure**

Tensor network methods are much more clever. Instead of storing the entire quantum state, we break it up into smaller pieces—tensors—that are linked together, kind of like a network. The idea is that if the quantum system has low entanglement, only a tiny part of the full Hilbert space actually matters. In these cases, tensor networks allow us to simulate systems with up to a hundred qubits or more—far beyond what statevector methods can handle. However, if the entanglement grows rapidly, like in deep or highly connected circuits, the tensor network also blows up in size and becomes intractable.

**6 Summary: Where Classical Methods Fail**

In summary, both statevector and tensor network methods have serious limits. That’s what motivates the search for new methods, like the Pauli propagation approach I’ll discuss next.

**7 Schrödinger vs. Heisenberg: Two Pictures of Quantum Dynamics**

When we talk about quantum evolution, there are actually two equivalent ways to look at it: the Schrödinger picture and the Heisenberg picture. In the Schrödinger picture, the quantum state kei psi changes over time, while the observables stay fixed. In contrast, the Heisenberg picture does the opposite: the quantum state is kept fixed, and it’s the observables O\_H that evolve as time goes on. Both pictures lead to the same predictions for any measurement. But they offer very different ways to think about quantum dynamics, and, as we’ll see, the Heisenberg picture sometimes has major advantages for simulation.

**8 Why the Heisenberg Picture?**

So, why should we care about the Heisenberg picture? In most quantum experiments and simulations, we’re really just interested in the expectation value of some observable at the end—like measuring magnetization or energy. The Heisenberg picture lets us focus directly on the observable we care about without tracking the entire quantum state. This can be much more efficient: we only need to update the observable and then calculate its expectation value using the initial state, which is usually simple. So in many cases, especially for large or complicated systems, the Heisenberg picture can save us a huge amount of computational effort compared to simulating the whole state.

**9 Observables as Pauli Strings**

In quantum computing, the Pauli operator basis is basically the language that quantum circuits speak. Any observable acting on n qubits can always be written as a sum of Pauli strings(指). A Pauli string is just a tensor product of single-qubit I, X, Y, or Z operators, like X on qubit 0, I on qubit 1, Y on qubit 2, and so on. Importantly, each Pauli string has a property called its weight, which is just the number of qubits it acts on non-trivially. For example, the string X I Y has weight 2, since it’s non-identity on qubits 0 and 2. These Pauli strings form an orthonormal basis for all n qubit operators. What’s really nice is that, as a quantum circuit acts, each Pauli string evolves into a new combination of Pauli strings, making this basis especially convenient for tracking observables through a circuit.

**10 Heisenberg Back-Propagation: The Key Idea**

To see how this works in practice, imagine our observable is just a single Pauli string to start with. In the Heisenberg picture, as we back-propagate it through the circuit—layer by layer—each quantum gate acts locally, potentially turning our single Pauli string into a sum of several new Pauli strings. For example, after passing through the last layer, say layer L, it might split into two terms: one with weight 1, and one with weight 2. ~~As we continue propagating backward, maybe after layer L-2 we get a term with weight 3. But here’s the trick: if we’re using weight truncation, we simply discard that weight-3 term and only keep the lower-weight terms. This approach keeps the complexity under control~~

**11 The Challenge: Exponential Growth of Pauli Terms**

As we start to back-propagate observables through a quantum circuit using the Pauli basis, we quickly run into a major challenge: exponential growth. Here’s the issue: every non-Clifford gate can take a single Pauli string and turn it into a sum of several new Pauli strings. For comparison, Clifford gates only ever map a single Pauli string to another single Pauli string. But once non-Clifford gates appear, each step can split a Pauli string into many more, and after a few layers these branching snowballs into an exponential number of terms. Storing all these terms quickly becomes just as hard as simulating the entire quantum state directly. That’s why, in practice, we have to keep only the most important Pauli terms and safely throw away the rest.

**12 Pauli Propagation Algorithm: Truncation in Action**

To deal with the exponential growth of Pauli terms, the Pauli propagation algorithm uses truncation. Instead of keeping all possible Pauli strings, we only keep those with weight less than or equal to some cutoff k. For each layer, we **apply the gate to get a new sum of Pauli strings, then immediately truncate, keeping only the strings with weight up to k (指向公式)**. Then we move to the next layer—again, apply the gate, expand, and truncate. This repeat-until-done approach ensures that, even after all the layers, we’re left with an efficiently represented observable and can compute its expectation value on the initial state（指向公式）. The key point is that by truncating at every step, we prevent the number of terms from exploding, while still keeping the answer accurate.

**13 Why Does Pauli Propagation Work?**

Why does truncation actually work? The real guarantee comes from a rigorous theorem—but only for locally scrambling circuits. In these circuits, every layer spreads local information so efficiently that even a simple observable becomes highly complex as it propagates. The theorem tells us that the mean squared error we make by truncating at weight k drops off exponentially as k increases. In other words, the total impact of all the Pauli strings we throw away is tightly bounded.

**14 Theorem: Efficient Simulation with Pauli Propagation**

With all this in mind, there’s another very powerful theorem behind Pauli propagation. It tells us that if the circuit is locally scrambling, we can achieve any desired accuracy and confidence just by choosing the right truncation weight k. The runtime only grows polynomially with the system size n and the desired precision—so we’re no longer facing exponential blowup for these hard circuits. Even better, if each layer only acts on a small number of qubits, things get even faster.

**15Motivation: Concrete Error for a Given Circuit**

So far, the theorems we’ve talked about give us guarantees on average—over random, locally scrambling circuits. But in a real experiment, we’re usually interested in one specific quantum circuit, not an average over many. For any particular circuit, the actual truncation error can be much smaller than the worst-case bound from theory. This is where Monte Carlo sampling comes in. By randomly sampling different Pauli propagation paths, we can directly estimate the truncation error for our specific circuit. This gives us an unbiased, practical measure of how accurate our simulation really is in the cases we care about.

**16 Monte Carlo Estimation: Sampling Pauli Paths**

To estimate the true error for a specific quantum circuit, we use Monte Carlo sampling of Pauli paths. See the figure on the right, suppose our observable is just a single Pauli string to start with. After propagating it backward through the last layer, it might split into a sum of three new Pauli strings（指）. According to the squared magnitude of each coefficient, we randomly pick one of them. Then, we take that chosen Pauli string and pass it through the previous layer, where it might split again—say, into two string—and again we sample one of the two based on the coefficients. We keep repeating this, sampling one Pauli string per layer, all the way back until we reach s\_0. This full sequence (鼠标移动) of sampled strings is called a Pauli path. By repeating this process many times, we can aggregate the results to get an unbiased estimate of the truncation error for the specific circuit we care about.

**~~17 Monte Carlo Sampling Estimators~~**

~~To actually get a quantitative estimate of the truncation error using Monte Carlo, here’s what we do in practice. For each Pauli path we sample, we check whether it ever produced a Pauli string with weight greater than our cutoff k. If it did, we calculate its contribution using a normalization factor and the overlap with the initial state【指向公式】. If the path always stayed below or at the cutoff, its contribution is zero. Once we’ve collected a batch of N samples, we simply average their contributions together to estimate the mean squared error for this particular circuit 【指向公式】. We can also estimate the statistical variance of this error estimate by looking at how much these individual sample values fluctuate around the average 【指向公式】. This gives us a practical and unbiased way to certify, for any specific quantum circuit, exactly how reliable our simulation result is.~~

**18 Certified Error Estimate: Monte Carlo Guarantees**

The final piece is to certify that our Monte Carlo estimate for the error is actually reliable. For any observable and any locally scrambling circuit, there’s a strong mathematical guarantee: if we take enough independent Pauli path samples—specifically, a number that scales with the number of circuit layers and the desired accuracy—then with high probability, our error estimate will be within any target precision we set. This means that not only do we get an unbiased estimate, but we can also rigorously control both the accuracy and confidence level of our result, just by choosing the number of samples appropriately.

**19 Implementation Highlights: Bit-Mask Encoding**

One of the key engineering tricks behind Pauli propagation is how we encode Pauli strings. Instead of storing each operator separately, we represent every n-qubit Pauli string as a single 2n-bit integer. The lower n bits indicate which qubits have an X component, while the upper n bits indicate where there’s a Z component. Let me break down using an example, say we have n = 3 qubits, take the Pauli string Y on qubit 2, I on qubit 1, and X on qubit 0—in Qiskit’s little-endian convention, that’s Y₂ I₁ X₀. We encode this as a 6-bit integer: the lower three bits (bits 0, 1, 2) are for X, and the upper three are for Z.

Here, X₀ and Y₂ both have an X component（指向表格）, so lower bits 0 and 2 are set: x = 101（指）. Y₂ also has a Z component, so upper bit 2 is set: z = 100（指）. Combine these to get 100101 in binary（指）, or 37 in decimal.

Now, if you look at the table below, you can see Y is encoded by having both the corresponding X and Z bits set for that qubit.

Quantum gates then just become fast binary bitwise operations on these integers. This compact encoding makes every gate update efficient in practice.

**20 Benchmark: Staircase Random SU(4) Circuit**

We benchmark Pauli propagation on this 2D staircase random SU(4) circuit because it’s an exceptionally hard challenge for any classical simulator. With 36 qubits, the system is already beyond the reach of statevector simulation on ordinary supercomputers, and even advanced tensor network methods fail for this level of entanglement. Each SU4 gate is a generic, maximally entangling two-qubit operation, which means we’re really challenging the algorithm. The circuit layers are arranged in a staircase pattern, which makes it locally scrambling and quickly spreads information, leading to strong global entanglement.

Now, if you look at the figure on the right, each grid point is a qubit, and every blue block is a two-qubit SU(4) gate. The lighter blocks are applied earlier in the circuit(指), the darker ones later（指）. Importantly, this figure just shows a single “repetition,”, which actually means 60 layers—not one layer.

For the benchmarking experiments, we vary the number of circuit repetitions from 1 to 5. The observable we focus on is Z\_36（指）, which measures the last qubit, and the initial state is just all zeroes.

**21 Why KAK Decomposition?**

When running real quantum circuits, the hardware can’t actually implement an arbitrary SU(4) two-qubit gate directly—it only supports a small set of native gates. That’s where KAK decomposition comes in. The KAK decomposition lets us take any SU(4) gate and break it down into a sequence of standard operations: specifically, three entangling gates—like RXX, RYY, and RZZ—along with twelve single-qubit Euler rotations. This process ensures we can map every random SU(4) gate in our benchmark onto real hardware. The figure illustrates how every SU(4) is decomposd into these basic building blocks.

**22 Truncation: Theoretical Bound vs Practice**

In theory, the error bounds are guaranteed if we truncate only after each SU(4) block（指）. But in practice, we actually truncate after every single decomposed gate—like each RXX, RYY, or Euler rotation（指）. This extra truncation keeps the number of Pauli terms from exploding, which dramatically speeds up the simulation, while still maintaining high accuracy.

**23 Truncation Weight k: Runtime Scaling**

A key feature of Pauli propagation is how runtime scales with the truncation weight k and circuit repetition. As you can see on the right, runtime grows exponentially with k. But the scaling with circuit repetitions L is only polynomial. So, setting k is a tradeoff between accuracy and speed.

**24 Monte Carlo: Pauli Weight Distribution**

One useful thing we can learn from Monte Carlo sampling is how entangled the system gets as the circuit evolves. Each sample gives us a final Pauli string, s\_0, and if we look at the histogram of the weights of these strings—shown in the plot on the right—we see a clear trend(指): as the number of repetitions increases, the typical weight of s\_0 rapidly rises. This means the evolved observable becomes more and more global, spreading over many qubits

**25Truncation MSE: Empirical vs. Theoretical Bound**

Here we’re comparing the actual truncation error, estimated using Monte Carlo sampling, to the theoretical upper bound from the theorem. The plot on the right shows that the real error—those solid points(指)—is always much smaller than the analytic worst-case bound, which is shown as the dashed line（指）. You can also see that as we increase the truncation weight k, the error decays. So in practice, the method is not only theoretically justified, but even more accurate than the theory might suggest for realistic circuits.

**26 Kicked Transverse-Field Ising Model (TFI) I**

Now that we’ve finished discussing the SU(4) staircase benchmark, let’s move on to the kicked transverse-field Ising model, or TFI. In this model, we have n qubits with spins arranged on a two-dimensional lattice. The Hamiltonian has two key parts: an Ising interaction between neighboring spins（指）, which encourages them to align along the z-direction, and a transverse field along the x-direction, which induces quantum fluctuations—h\_x is the strength of this field. Each spin interacts only with its neighbors, forming a grid.

What makes this model particularly challenging is that the Ising and transverse field terms don’t commute, so the quantum dynamics become highly nontrivial.

**28 TFI Dynamics and Trotterization**

The exact evolution can’t be implemented natively on a quantum computer, since the interaction and field terms don’t commute. So, to simulate the dynamics of the TFI model on a quantum circuit, we use Trotterization to break up the evolution into discrete time steps instead. Each time step alternates two layers: a layer of parallel rx gates—which are just single-qubit rotations representing the transverse field(指)—and a layer of parallel rzz gates（指）, which are two-qubit entangling gates capturing the Ising interactions. By repeating this rx,rzz pattern many times across the 2D lattice, we mimic the complex time evolution of a TFI Hamiltonian using only local quantum gates.

**29 IBM Eagle Heavy-Hex Topology**

Now let’s look at the IBM Eagle quantum processor, which we use for experimental benchmarking. Eagle is a 127-qubit chip where the qubits are arranged in a heavy-hex lattice. In this layout, each node is a physical qubit, and the edges indicate which pairs of qubits are directly connected by two-qubit gates. The figure on the right shows the actual heavy-hex connectivity map for the Eagle device.

**30 Experimental Protocal: Kicked Ising Dynamics**

Here’s how the experiment is set up on the IBM Eagle chip—and we follow the same protocol with Pauli propagation. We start in the all-zero state and measure Z62, which is the central qubit(指). The circuit runs for 20 Trotter steps. In each step, we first apply Rx gates to every qubit, sweeping the field strength, and then RZZ gates to all nearest-neighbor pairs according to the heavy-hex connectivity. Because of hardware constraints, these two-qubit gates are scheduled in three parallel, non-overlapping layers, as illustrated by the colored edges in the figure on the right.

**31 Results: Classical vs. Quantum Experiment**

This is the result comparing classical simulation to the IBM Eagle experiment. On the x-axis, we have the transverse field strength parameter theta\_h（指）, and on the y-axis is the expectation value of Z62 after 20 Trotter steps. The blue points are experimental data from Eagle, already error-mitigated(指). The magenta dashed curve shows Matrix Product State (MPS) simulation, which starts to fail as entanglement grows（指）. The solid lines are our Pauli propagation results at different truncation weights k. At k=7, Pauli propagation is about 90 times faster than MPS and remains more accurate. Even though this circuit isn’t locally scrambling, Pauli propagation still agrees well with real quantum computer results. Belief propagation tensor network can reach similar accuracy, but require much more customization—whereas Pauli propagation is broadly applicable.

**32 Further Research Directions**

Looking ahead, there are two main directions for expanding Pauli propagation. First is apply it in quantum optimization. Here, fast estimation of cost functions can really speed up parameter sweeps in variational algorithms, which are widely used in quantum chemistry, finance, and logistics. Second is benchmarking noisy quantum devices. By inserting realistic noise models—like depolarizing or amplitude damping channels—between gates, Pauli propagation lets us predict hardware performance and evaluate error mitigation strategies.

**第33 Conclusion**

To wrap up, we’ve shown that Pauli propagation with low-weight truncation allows us to estimate observables in circuits that are beyond the reach of statevector and tensor network methods. In practice, the actual Monte Carlo error is much smaller than the theoretical upper bound, even when we truncate more aggressively than the theoretical prerequisite. On real hardware benchmarks—like the IBM Eagle kicked Ising experiment—Pauli propagation matches experimental results with moderate truncation, and outperforms MPS both in speed and accuracy. While belief propagation tensor network methods can also reach high accuracy, they require custom modifications for device topology. By contrast, Pauli propagation is completely general, with no structural assumptions. Most importantly, this strong performance holds even for circuits that aren’t locally scrambling, which highlights the broad power and flexibility of this approach.

**Q and A**

问题 1：“宏观动机”

“你能用简单的语言再给我解释一下，为什么模拟量子计算机从根本上就这么困难？我们为什么不能直接用一台强大的超级计算机来做呢？”

Sure. The core challenge is the exponential growth of quantum state space: an nnn-qubit system requires tracking 2n2^n2n complex numbers. For just 35 qubits, this exceeds the memory of a supercomputers. Traditional simulation (Schrödinger picture) tracks this entire state. My work instead follows only the evolution of specific observables (Heisenberg picture), which is far more efficient when we only care about a few measurement outcomes.

问题 2：你方法的核心思想

“你提到了‘泡利传播’和‘截断’。你能给我一个直观的解释吗？这听起来像是在丢弃信息，那为什么结果还能保持准确呢？”

Certainly. When we evolve an observable backward through a quantum circuit, a simple Pauli string—like an ‘X’ on one qubit—spreads into a sum of many, increasingly complex Pauli strings. This is “propagation,” and their number grows exponentially.

“Truncation” addresses this: in chaotic circuits, the most complex, high-weight Pauli strings contribute negligibly to measurements—their effect is exponentially suppressed. So, we safely discard all Pauli strings above a certain weight threshold kkk. As shown in my results (e.g., Slide 25), this controlled approximation introduces only a tiny error, which shrinks exponentially as kkk increases, making the simulation tractable.

问题 3：理解你的结果

“在第30页的PPT上，你展示了你的模拟结果（实线）和真实的IBM量子计算机实验数据非常吻合。我们应该从这张图得出什么最重要的结论？”  
The key takeaway is that state-of-the-art classical simulation methods remain highly competitive. My Pauli propagation approach (solid line) closely matches IBM’s 127-qubit experimental data and outperforms leading tensor network methods (MPS, magenta dashed line), especially as entanglement increases. It also runs about 90 times faster.

This shows that, for certain observables, classical simulation is still comparable with today’s quantum hardware—raising the bar for “quantum advantage.” However, it’s important to note: if the observable itself is a high-weight Pauli string, this method becomes inefficient and loses its edge.

问题 4：澄清“置乱”的概念

“你提到‘局域置乱线路’对你的理论很重要。这在物理上意味着什么？”

Physically, a “scrambling” circuit spreads local information quickly across the whole system—like dropping ink into a blender instead of still water. After scrambling, the effect of a local observable gets mixed into all qubits and becomes extremely complex. This rapid mixing is what justifies our truncation: information gets hidden in high-weight Pauli terms, which can then be safely ignored.

**问题 5：关于方法的选择**

“你提到了几种不同的经典模拟方法，比如你自己的泡利传播法，还有态矢量法和张量网络法。听起来它们各有优劣。在实际研究中，一个科学家应该如何选择使用哪种方法？有没有一个简单的判断标准？”

Here’s a simple rule of thumb:

* For small systems (say, n < 30 qubits), use statevector simulation—it’s exact and efficient.
* For larger systems with low entanglement (like 1D chains), tensor network methods (e.g., MPS) work best—they scale well as long as entanglement remains limited.
* For large, highly entangled, or scrambling systems—especially when your observable is a low-weight Pauli string—Pauli propagation is preferred. It tracks only the observable’s evolution and efficiently approximates the result by truncating high-weight terms.

**问题 6：关于量子计算机的价值**

“既然你的经典模拟在127比特的问题上比真实的量子计算机做得又快又好，这是否意味着量子计算的炒作成分太大了？如果经典方法还能不断进步，那么我们为什么还需要投入巨资去建造这些近期的量子设备呢？”

What my results show is that, for some important problems, classical simulation is still catching up and can set a moving target for quantum hardware. But we know that, fundamentally, classical methods always hit a wall: as system size, circuit depth, or observable complexity increases, the cost eventually grows out of reach. Quantum computers are designed to scale past that wall. So while classical methods keep raising the bar, quantum hardware is the only path to truly solving problems that are impossible for any classical supercomputer—especially as devices improve and error rates drop. The “race” between classical and quantum is what drives progress on both sides.

来自你的导师（Prof. Hamza Fawzi，专家）的问题

问题 1：KAK分解与截断

“Steven，在第22页PPT中你讨论了KAK分解。我们讨论过的误差上界定理，是在对一个完整的、局域置乱的SU(4)模块进行一次截断时才成立的。但在你的实现中，你是在分解后的*每一个基本门*之后都进行了截断。你如何为这个选择辩护？它在理论和实践上有什么影响？”

Theoretically, the error bound holds only if we truncate after a full, locally scrambling SU(4) block—not after each basic gate. But in practice, I follow a heuristic: truncating after every gate, just like the original paper does. This is mainly to control the combinatorial explosion in Pauli terms and keep the algorithm efficient. While this means we lose the strict theoretical guarantee, the empirical errors (see Fig. 10, p.25) are still much lower than the formal bound—so for random circuits, this approach is both practical and safe.

问题 3：与其他先进方法的比较

“你在TFI模型上的结果令人印象深刻。然而，最近关于信念传播张量网络（belief-propagation tensor networks）的工作也显示出在这个完全相同的问题上表现出色。你声称泡利传播更‘通用’。你能否详细说明一个具体的场景或线路类型，在那种情况下，你预期泡利传播会决定性地超越BP-TN？”

BP-TN is very strong when your circuit always connects the same pairs of qubits in a regular pattern, like a grid or fixed layout.

But if your circuit **changes the connections between qubits in every layer**, or **lets any qubit interact with any other**, then BP-TN loses its efficiency.

Pauli propagation, on the other hand, works equally well no matter how the connections change.

**问题 9：关于实现的性能权衡**

“在你的 pauli\_term.py 实现中，你选择将X和Z掩码打包进一个2n比特的整数。你提到当n>64时，使用两个整数的方案可能更好。你能详细阐述一下这里的性能权衡吗？单整数方案会引入什么特定的计算开销，使得它在n非常大时变得不适用？”

If n is small (say, ≤64), packing X and Z into a single integer is very memory efficient, but extracting the Z part for the i-th qubit is slightly more awkward, since it’s stored after all the X bits.

If n is large, this single integer gets replaced by a slower, arbitrary-precision object, and bit operations become inefficient.

In that case, using two separate bitmasks for X and Z is clearer and faster, though it uses more memory.

**问题 10：对TFI模型结果的深入思考**

“TFI模型上的成功尤其有趣，因为正如你所说，这个线路并*不是*从局域置乱系综中抽取的——RZZ门的旋转角是固定的。泡利传播的理论保证是基于置乱特性的。你认为为什么这个方法在这种非置乱的场景下依然表现如此出色？TFI动力学的背后，是什么样的物理或统计特性，即使没有真正的随机门，也导致了高权重泡利项被抑制？”

Even though the RZZ angle is fixed, the kicked-Ising circuit still acts like a **randomly scrambling system** in terms of operator dynamics:  
Each RX rotates every spin, and each RZZ step rapidly spreads this effect to neighboring sites.  
As a result, a local Z operator quickly spreads out, and the amplitudes for high-weight Pauli strings drop off exponentially—just like in a genuinely randomly scrambling circuit.

So, even without true locally scrambling circuit, truncating high-weight strings discards almost no useful information.