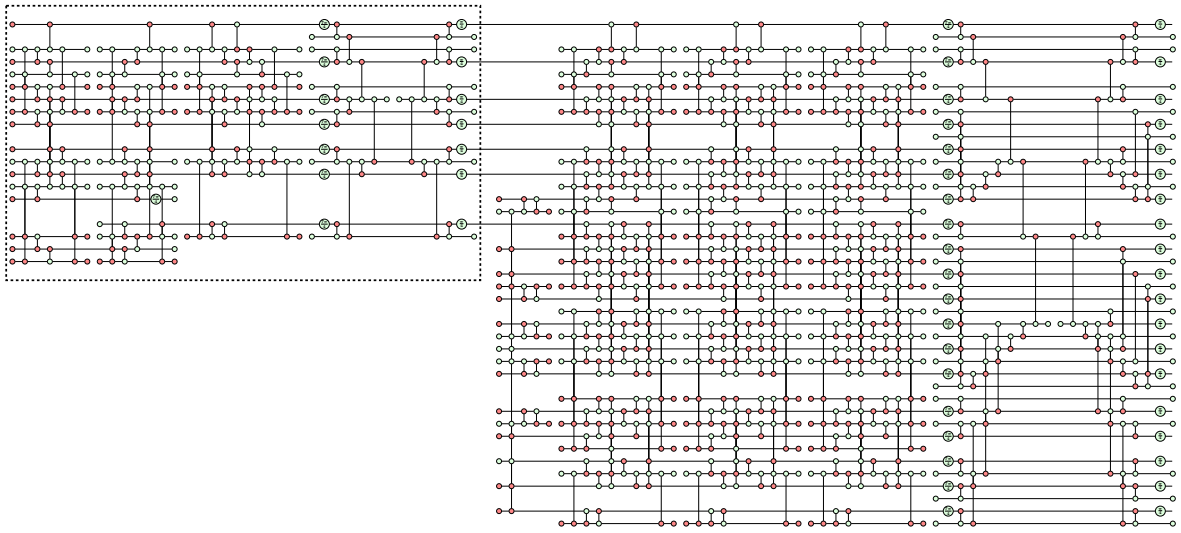
Simulating magic state cultivation with few Clifford terms

Kwok Ho Wan[1] and Zhenghao Zhong[2]

[1]Blackett Laboratory, Imperial College London, South Kensington, London SW7 2AZ, UK
[2]Mathematical Institute, University of Oxford, Andrew Wiles Building, Woodstock Road, Oxford OX2 6GG, UK
February 18, 2026

Building upon [arXiv:2509.01224], we present a few methods on how to simulate the non-Clifford $d = 5$ magic state cultivation circuits [arXiv:2409.17595] with a sum of ≈ 8 Clifford ZX-diagrams on average, at 0.1% noise. Compared to a magic cat state stabiliser decomposition of all 53 non-Clifford spiders (6,377,292 terms required), this is more than 7×10^5 times reduction in the number of terms. Our stabiliser decomposition has the advantage of representing the final non-Clifford state (in light of circuit errors) as a sum of Clifford ZX-diagrams. This will be useful in simulating the escape stage of magic state cultivation, where one needs to port the resultant state of cultivation into a larger Clifford circuit with many more qubits. Still, it's necessary to only track ≈ 8 Clifford terms. Our result sheds light on the simulability of operationally relevant, high T-count quantum circuits with some internal structure.

Finally, we provide numerical results for full non-Clifford stabiliser rank simulation based on `tsim` along with optimisations using our cutting decompositions. Nearly 4×10^6 shots per second can be obtained on a laptop for the smaller $d = 3$ circuits at SD6 circuit level noise $p = 0.0005$, making it only ~ 1.1 times slower than its (circuit-unspecific and un-optimised) fully Clifford proxy simulation via `stim` using S gates.



(1)

1 Introduction

The $d = 5$ magic state cultivation circuit [1], shown in (1), is difficult to simulate due to its high T-count of 53 $T(T^\dagger)$ gates. The authors of [1] simulated most of the larger $d = 5$ cir-

Kwok Ho Wan: ((initials))1496((at))((9.81))mail.com, current affiliation: PsiQuantum, 700 Hansen Way, Palo Alto

Zhenghao Zhong: All authors contributed equally and are listed in an alphabetical order.

cuits by replacing $T(T^\dagger)$ gates entirely with $S(S^\dagger)$ gates. Applying the same replacement to the $d = 3$ sub-circuit (inside the black dashed box of (1)) reveals an $\approx 2\times$ discrepancy in logical error rate (LER) when benchmarked against brute-force state vector simulation. This demonstrates the need for simulation techniques that handle the non-Clifford gates exactly while remaining compatible with Monte Carlo error sampling.

In this work, we use techniques from [2] to decompose the $d = 5$ magic state cultivation circuit from (1) into ~ 8 Clifford ZX-diagrams on average, while fully accounting for Pauli errors on every edge in the ZX-diagram at error rates in the operationally relevant regime of 10^{-4} to 10^{-3}. A meaningful benchmark of logical performance[1] without the $T(T^\dagger) \to S(S^\dagger)$ substitution may therefore be feasible, given sufficient integration of ZX-calculus-based stabiliser decomposition into tools such as `tsim` [3].

For the smaller $d = 3$ sub-circuit, we complement the analytical decomposition with a full numerical simulation pipeline. By combining spider cutting with BSS stabiliser decomposition, the circuit compiles into 120 Clifford graphs with a total of 28 node terms, 192 sign terms, and 1,408 phase-pair terms. We implement several optimisations on top of `tsim` [3]: BLAS-accelerated evaluation via matmul-based binary row sums, enumeration-based sampling over the 32 measurement-outcome combinations, noiseless outcome caching, and full-program JIT compilation.

We further accelerate sampling by replacing per-shot channel sampling with a sparse geometric-skip sampler that generates events only at non-identity fire positions, deduplicating repeated noise-channel patterns with a persistent cross-batch evaluation cache, and compiling the combo-evaluation loop via `jax.lax.scan`. Together, these yield a throughput exceeding 4 million shots per second on a laptop (Apple M4 Macbook Pro) at circuit-level noise $p = 5 \times 10^{-4}$, only $\sim 1.1\times$ slower than a fully Clifford proxy simulation via `stim`, partially addressing the open problem of simulating magic state cultivation on common day-to-day hardware posed by [1]. Note that our pipeline is optimised specifically for this circuit, while `stim` is a general-purpose Clifford simulator. This confirms a logical error rate of $\sim 10^{-6}$ at $p = 10^{-3}$ for the $d = 3$ circuit without resorting to the $T \to S$ substitution or statevector simulations.

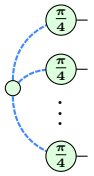
The following section presents the magic cat state decomposition as a baseline, after which we outline numerical evidence for the simulability of the $d = 5$ (and $d = 3$) cultivation circuits.

2 Magic cat state stabiliser decomposition

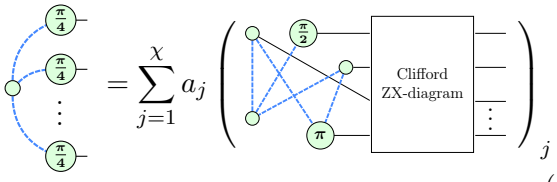
Magic cat states [4–6] are a family of states parametrised by integer m:

$$|\mathrm{cat}_m\rangle = \frac{1}{\sqrt{2}}(I^{\otimes m} + Z^{\otimes m})\,|T\rangle^{\otimes m} \quad , \tag{2}$$

this state can be written as the following ZX-diagram

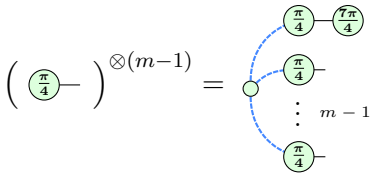
$|\mathrm{cat}_m\rangle =$  , (3)

with m legs. The magic cat state stabiliser decomposition is the expansion of this non-Clifford ZX-diagram as a sum of Clifford ZX-diagrams.



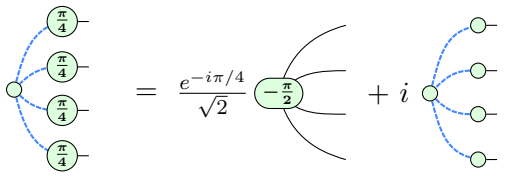
(4)

Explicit number of terms (χ) involved in this sum for various $|\mathrm{cat}_m\rangle$ is available at [5] Note that $|T\rangle^{\otimes(m-1)}$ can be obtained via a $|\mathrm{cat}_m\rangle$ state by measuring one of its legs by fusing it with a  :



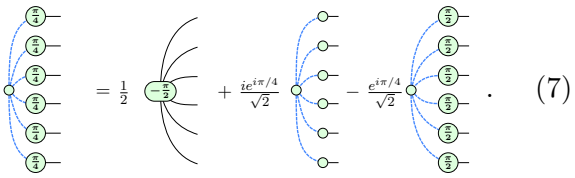
, (5)

we will call this the magic cat state stabiliser decomposition of $|T\rangle^{\otimes(m-1)}$. See examples of $|\mathrm{cat}_4\rangle$ and $|\mathrm{cat}_6\rangle$ [6][2]:

 , (6)

[1] And any extension to simulate the escape stage end-to-end.

[2] Tikz figures (6) and (7) taken from [6].

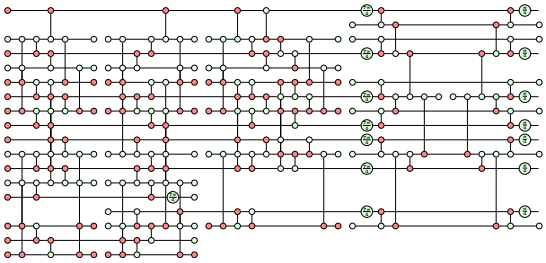


(7)

These will be used to decompose the $d = 3$ magic state cultivation circuit into Clifford terms.

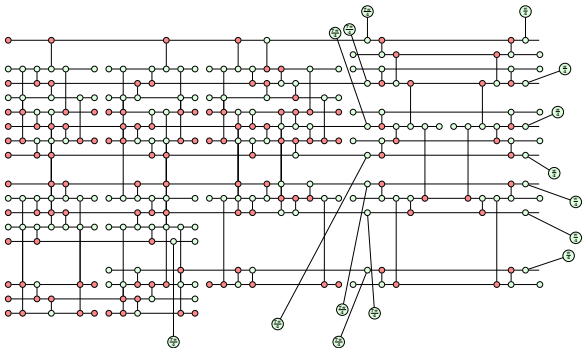
3 $d = 3$ magic state cultivation circuit

Now, let's see how one can use the magic cat state stabiliser decomposition [5] to strongly simulate the $d = 3$ magic state cultivation circuit ((8) or ZX-diagram inside the black dashed box of (1)) exactly. We first un-fuse all the $\frac{\pi}{4}$ / $\frac{7\pi}{4}$ of



, (8)

resulting in:



. (9)

Here, we counted 15 $\frac{\pi}{4}$ / $\frac{7\pi}{4}$ spiders. We can chop these $\frac{\pi}{4}$ / $\frac{7\pi}{4}$ spiders off and fuse them with the legs of the magic cat stabiliser decomposition of $|T\rangle^{\otimes 15}$ up to some local Clifford ($S = \mathrm{diag}(1, i)$) transformations on the 8 $\frac{7\pi}{4}$ spiders. The underlying $|\mathrm{cat}_{16}\rangle$ decomposition uses the $|\mathrm{cat}_4\rangle \otimes |\mathrm{cat}_6\rangle^{\otimes 3}$ decompositions fused together at some legs to obtain $|\mathrm{cat}_{16}\rangle$.

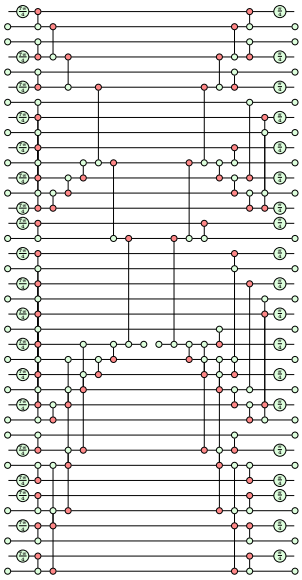
Using the table from [5, Table A.4], we see that this sub-circuit can be represented with $2 \times 54 = 108$ pure Clifford ZX-diagrams, whilst still maintaining its circuit structure without any ZX-rewrite/reduction/contractions. Hence, one can still insert errors in the same locations in each expanded terms circuit for Monte-Carlo simulations. This is useful in the context of computing logical error rates via sampling. Note that the number of terms here are double that of those in the table of [5] since we are counting pure Clifford ZX-diagrams, whereas each term in [5] has a single $\frac{7\pi}{4}$ spider. Perhaps more theoretical insights regarding the logical error rate deviation observed in [1] can be gained from looking at how error propagates across terms in the stabiliser decomposition of this circuit. This is left for future work.

This stabiliser decomposition of $|T\rangle^{\otimes 15}$ with an additional overhead of 108×, can be similarly applied to the 15-to-1 magic state distillation circuit [7] with encoded surface code patches [8] for example.

4 $d = 5$ circuit via magic cat state stabiliser decomposition

The $d = 5$ magic state cultivation circuit is just the $d = 3$ variant with additional Clifford gates followed by a larger double checking circuit at the end (shown in (10), see also appendix C for an extended discussion). This sub-routine has 38 $\frac{\pi}{4}$ / $\frac{7\pi}{4}$ spiders. Naively, if we un-fuse all the spiders of the $d = 5$ double checking sub-circuit (from (10)) inside (1) and replace them with magic cat state stabiliser decomposition of $|T\rangle^{\otimes 38}$, we will arrive at $2 \times 39{,}366 = 78{,}732$ pure Clifford terms [5]. Taking into account the $d = 3$ magic state cultivation circuit prior to the $d = 5$ double checking circuit, the full $d = 5$ circuit (in (1)) will have $108 \times 78{,}732 = 8{,}503{,}056$ terms.

Looking at the $d = 5$ circuit as a whole, with all the 53 total non-Clifford spiders, we can actually break this down into $|\mathrm{cat}_{17}\rangle$ and $|\mathrm{cat}_{38}\rangle$, converted to $|T\rangle^{\otimes 16} \otimes |T\rangle^{\otimes 37}$ via fusing each magic cat state to $\frac{7\pi}{4}$ spiders. This implies a lower number: 6,377,292 total pure Clifford terms. We emphasise that we need to simulate each ZX-reduced ZX-diagram with random Pauli-X/Y/Z errors on each edge, in the context of Monte Carlo sampling for computing logical error rates.

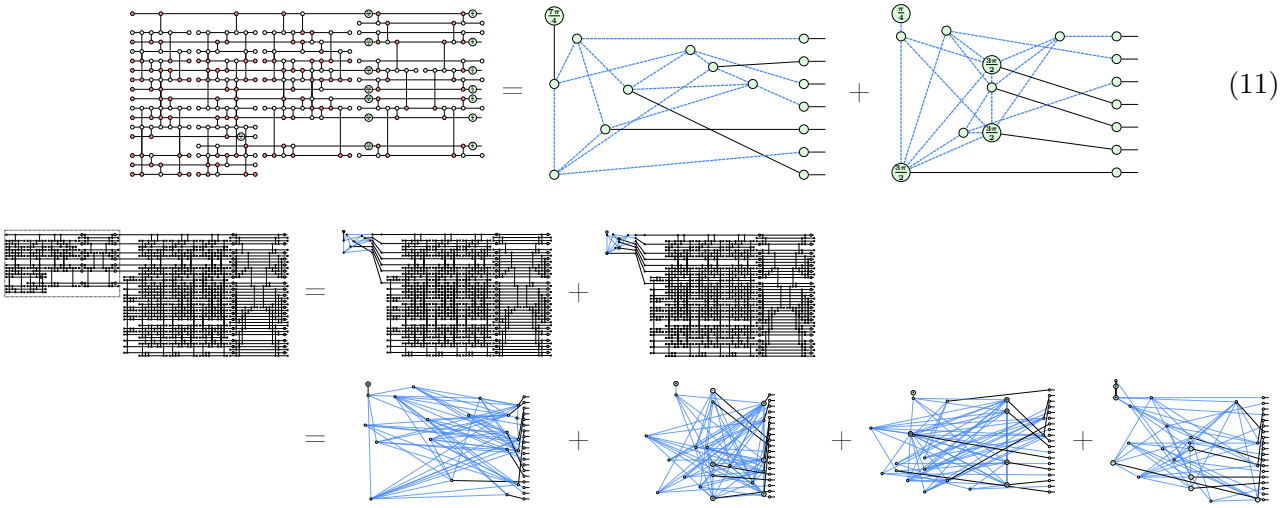


(10)

Simulating the $d = 5$ cultivation circuit in this manner would require 6 to 8 million additional terms per shot compared to an equivalent stabiliser circuit[3]. The only advantage is the inclusion of Pauli/Clifford errors comes at no additional cost, as they do not increase the number of terms in the stabiliser decomposition. In the next section, we will show how to reduce this massive overhead with the cutting decomposition even when Pauli errors are taken into account.

5 The cutting decomposition re-visited

In an earlier manuscript [2] we studied the cutting decomposition [5, 9, 10] and showed that the error-free $d = 3$ and $d = 5$ magic cultivation circuits can be represented with a stabiliser decomposition of 4 and 8 terms respectively [2]. We summarised the results below in (11) ($d = 3$) and (12) ($d = 5$).



(11)

(12)

A natural question to ask is whether this number of terms persists when random errors are applied to each edge in the ZX-diagram. For a feasible simulation, we require that Pauli errors be included on each edge of the ZX-diagram while still keeping the number of terms in its cutting decomposition manageable. Furthermore, how does the number of terms scale with the error rate?

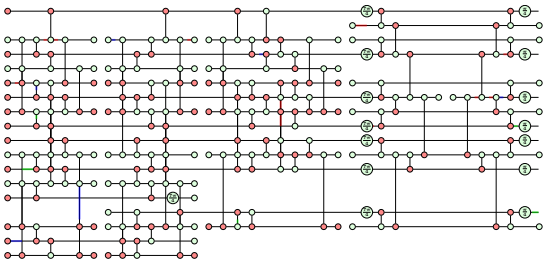
In order to study this, we subject every edge of the cultivation circuits to Pauli-X/Y/Z error, where each Pauli-X/Y/Z error occurs with probability $p/3$, governed by the channel:

$$\mathcal{E}(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z)\ . \quad (13)$$

This is the standard single qubit depolarising error channel applied to every edge. For example, in the $d = 3$ circuit (from (8)), an error realisation

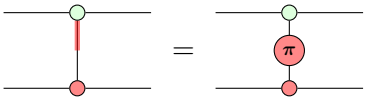
[3] Swapping $T \to S$.

is shown in (14),



, (14)

where coloured half-edges are used to mark Pauli errors on its ZX-diagram edges. This edge decoration notation is similar to Pauli webs [11] and is consistent with notations from `pyzx` [12]. Note that this error model has a higher error density in its ZX-diagram compared to the SD6 circuit-level noise model [13]. Errors happening 'in-between CNOTs' shown in (15):



(15)

are permissible in our error model.

We then applied the same cutting decomposition outlined in [2] to the Pauli-errored $d = 3$ and $d = 5$ cultivation ZX-diagrams. At the end of the cutting decomposition, if we failed to decompose terms into low T-count ZX-diagrams (whereby each term contains only 1 or less non-Clifford spiders), we further apply **secondary** magic cat state decomposition (outlined previously) to those terms. We noticed that in the operationally relevant error regimes of magic state cultivation - $\mathcal{O}(10^{-4})$ to 10^{-3} edge error rate, the number of terms is still very reasonable. On average, ~ 8 (~ 4) terms are needed for the $d = 5$ ($d = 3$) circuits in those error ranges (see figures 1 and 2). To account for any potential large deviation from the mean number of terms, albeit with low probability, we show the maximum number of terms in the same plots. In our numerical experiment, we observed no more than 432 (8) maximum number of terms in the same error intervals over 10^4 shots. There is a single shot containing 432 terms at 0.0015 error rate for the $d = 5$ circuit. We don't observe this in the two subsequent higher error data points, this is likely due to our sample size. Nonetheless, the average number of terms is still very low.

Based on these numerical results, we suggest that one can simulate the magic state cultivation circuits, post-selected upon all +1 measurements

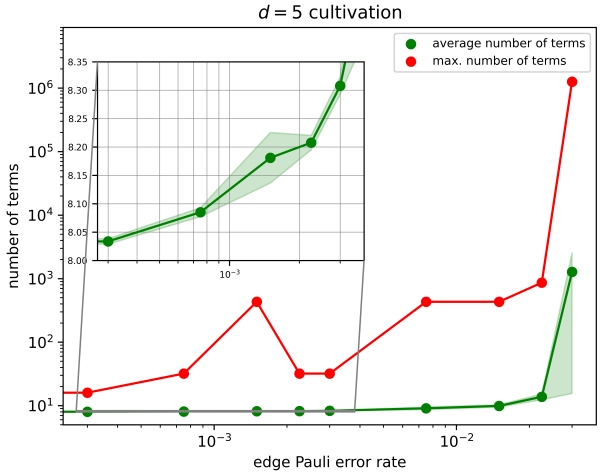


Figure 1: Average and maximum number of terms in the cutting stabiliser decomposition for the $d = 5$ circuit. There is a single event containing 432 terms at edge error rate of 0.0015.

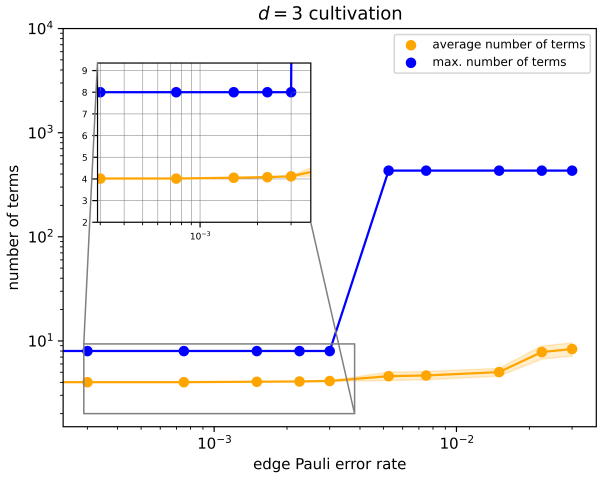


Figure 2: Average and maximum number of terms in the cutting stabiliser decomposition for the $d = 3$ circuit.

with the cutting stabiliser decomposition [2]. On average, this appears to add only a minor overhead on top of the stabiliser simulations. Next, we will consider the case where -1 parity measurement outcomes are permitted.

6 Randomised measurement spiders

Post-selecting upon $+1$ outcome exclusively is unrealistic in the context of magic state cultivation, since some parity measurement results may yield a -1 value and still correspond to an accepted shot provided the syndrome (products of parities) in the associated detecting region [14] returns a

+1 value. To address this issue[4], we present additional simulations in this section. Our simulation will focus on the $d = 5$ circuit, although the same reasoning can be applied to the lower T-count $d = 3$ circuit.

As an initial step, we repeated the simulations from the previous section for the $d = 5$ circuit, this time allowing each measurement spider to be randomly flipped with uniform probability. In the simulation, we effectively change all —● → —(a_i π) and —○ → —(a_j π), where $a_k \in \{0, 1\}$ are uniform random numbers representing potential measurement results. Our approach to relax the constraint of post-selecting on all +1 parities is to emulate it by introducing random flips on the measurement spiders.

We noticed a sharp increase in the mean and maximum number of terms required as shown in figure 3. This simulation proceeds by decomposing terms via magic cat state decomposition if the cutting decomposition failed to produce terms containing 1 or less —(π/4)— / —(7π/4)— spider. The results from figure 3 are concerning as early fault-tolerant quantum devices are expected to exhibit error rates of $\mathcal{O}(10^{-3})$ [15]. Confirming cultivation on such devices would require $\sim 10^2$ to 10^3 more terms per shot in a Monte-Carlo simulation, potentially making this stabiliser decomposition approach impractical.

If instead of using magic cat states in the secondary decomposition, we can,

1. perform a BSS decomposition[5] [16] iteratively on each surviving ZX-diagrams stemming from the cutting decompositions, and

2. perform more cuts in the first stage of cutting decompositions.

By combining these two strategies, we obtain significantly improved scaling of both average and maximum number of terms with respect to edge Pauli error rate, as shown in figure 4. The average number of terms stays near ≈ 8, even at higher error rates once intractable. The cutting decomposition appears to struggle when the ZX-diagram being decomposed contains a large number of spiders or errors. Hence it must be supplemented with additional cuts and/or BSS as a secondary step.

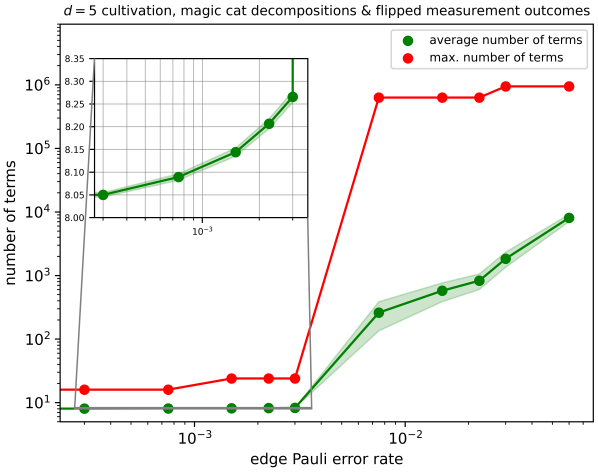


Figure 3: Average and maximum number of terms in the cutting stabiliser decomposition followed by magic cat secondary decompositions for the $d = 5$ circuit. Measurement flips are applied, flipping each measurement single leg spiders to $a\pi$, $a \in \{0, 1\}$ uniformly.

The results presented here are preliminary simulations. There are certainly opportunities to further optimise these decompositions and how to combine them appropriately. Potentially, one can incorporate the star edge decompositions from [17, 18] (see appendix D). See section 9 for an end-to-end $d = 3$ circuit simulation via integration of cutting with `tsim` [3].

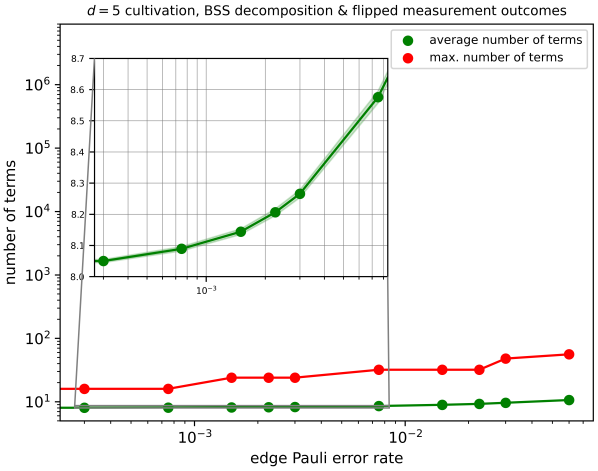


Figure 4: Average and maximum number of terms in the cutting stabiliser decomposition followed by BSS secondary decompositions for the $d = 5$ circuit (with measurement flips).

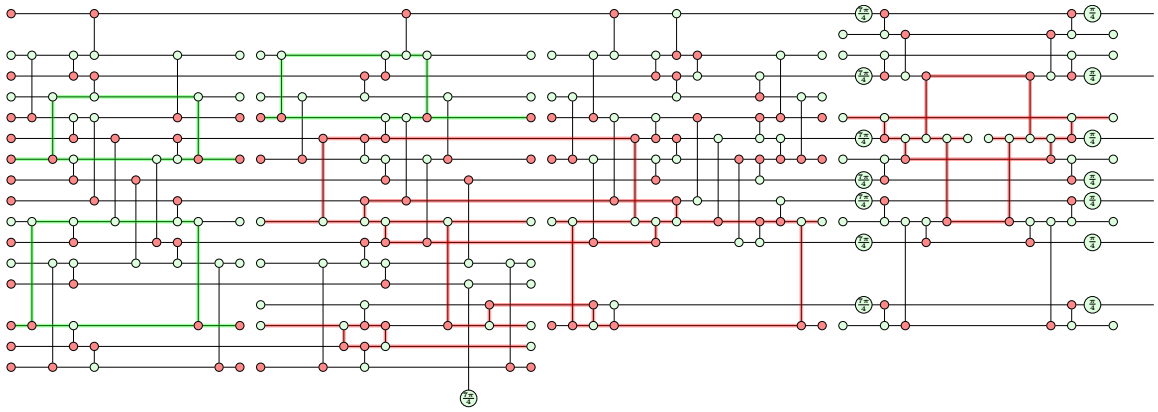
[4] On Scirate, the discussion platform for arXiv preprints (especially in quantum information sciences), user KdV raised concerns about the first version of our arXiv submission, in particular, our reliance on post-selection of +1 parity measurement results. See the full discussion at https://scirate.com/arxiv/2509.08658.

[5] With `pyzx.simulate.find_stabilizer_decomp(g)`.

7 Post-selected closed Pauli webs

Additional optimisation can be implemented such that stabiliser decomposition is applied to only a subset of all measurement outcomes. This will further reduce runtimes. Magic state cultivation is a post-selected protocol, one can determine which measurement outcomes are considered acceptable shots and should therefore be retained during simulation. This effectively corresponds to performing stabiliser decomposition on error realisations for which all detecting regions [11, 14] remain un-violated [14]. Six examples (3 red, 3 green) of these detecting regions/closed Pauli webs are illustrated in (16) for the $d = 3$ circuit.

It is not necessary to perform stabiliser decomposition to all possible shots as certain measurement patterns are rejected due to post-selection. For the $d = 3$ and $d = 5$ circuits with degenerate injection, we have translated all the detectors from [19] into closed Pauli webs. These are available in appendix F and G as figures and under the folders `d3_det_webs` and `d5_det_webs` as supplementary tikz files.



(16)

We have repeated the simulations from the previous sections, this time post-selecting on all the detecting regions and ensuring all the closed Pauli webs [11] (equivalent to detectors in [19] or detecting regions in [14]) remain un-violated, for the $d = 5$ circuit (see figure 5).

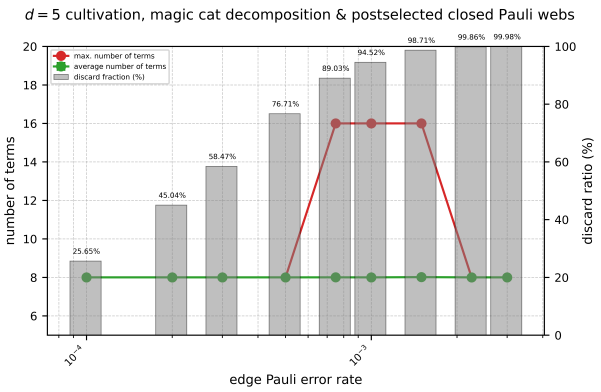


Figure 5: Average and maximum number of terms in the cutting stabiliser decomposition followed by magic cat secondary decompositions, for the $d = 5$ circuit. The error realisation were post-selected such that all detecting regions returned $+1$ parity. We also included the discard ratio at each error rate.

We observed a near constant average number of terms (≈ 8) across the relevant error regime of $\mathcal{O}(10^{-4})$ to $\mathcal{O}(10^{-3})$ when using only the magic cat state secondary decomposition. However, single occurrences of error realisations led to 16 terms at edge Pauli error rate of 0.00075, 0.001 and 0.0015 with 10^5 samples[6]. We also included the discard ratio for various error rates under our error model in the same figure (5). Note that our error model has a higher error density, so the discard ratio is expected to be higher compared to [1] as these correspond to different error models[7]. Figure 5 provides early indications that the realistic number of terms in simulations with post-selected detectors/closed Pauli webs may still remain very low.

[6] Likely due to small sample size.

[7] Note: $p_{\text{edge Pauli error}}$ here $\neq p_{\text{circuit-level noise}}$ from [1].

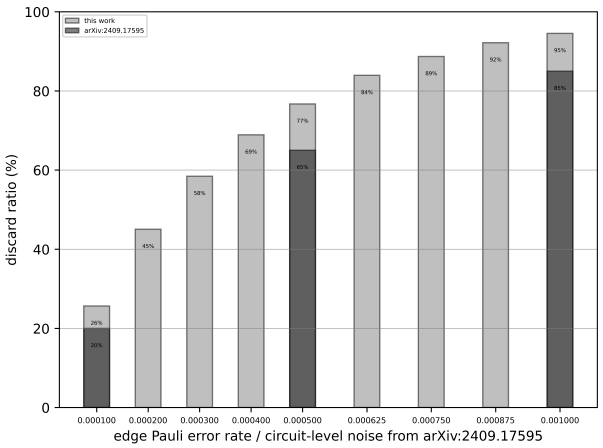


Figure 6: Comparison of discard ratios with results from [1].

To compare with the existing literature, figure 6 shows the discard ratio for the $d = 5$ circuit over a range of edge Pauli error rates, alongside the discard ratios reported in figure 2 of [1]. As anticipated, our error led to slightly higher discard ratio, in line with our initial expectations.

Overall, the numerical evidences from the past three sections indicate that, for simulations in the $\mathcal{O}(10^{-3})$ error regime, the average number of terms remains at ≈ 8 regardless of the chosen secondary decomposition, or if we post-select upon detector values.

8 A sketch on an end-to-end logical error rate simulation

In order to perform an end-to-end simulation of magic state cultivation, we present two approaches, which we shall name the *sample measurements* approach and *closed Pauli web post-selection* approach.

8.1 Sample measurements approach

In this first approach, we sample the measurement results explicitly. We can start by contracting/simplifying the errored[8] ZX-diagram up to a measurement layer, excluding the measurement spiders. Then perform a stabiliser decomposition on that sub-diagram up to the measurement time slice with measurement spiders removed, hence more open legs. A sum of stabiliser terms will be returned from the decomposition. Sample those terms (they are all Clifford) and combine them into actual measurement realisations. Then input the sampled measurement outcomes into the original sub-diagram, included as $n\pi$ ($n \in \{0, 1\}$) phases in the corresponding measurement spiders:  . Finally, conditioned on all un-violated detecting regions inferred from measured parities, proceeding to the next measurement stage and repeat this procedure. Please see an example of the sketch outlined applied to the initial stages of the $d = 3$ circuits in appendix H and the following pseudocode in algorithm 5 in appendix E. A drawback to this approach is that each shot may require multiple rounds of stabiliser decompositions proportional to the number of measurement layers. This will increase the actual run time.

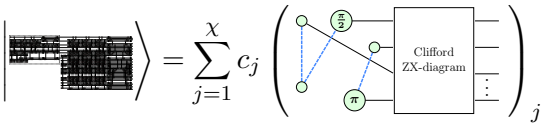
[8] With known error locations and types.

8.2 Closed Pauli web post-selection approach

In this next (better) approach, we can post-select on closed Pauli webs without ever generating measurement values. To run and simulate a numerical experiment with errors, Pauli errors must first be initialised in the ZX-diagram, with their error types and locations stored. Using this information about the errors, one can form parity values of closed Pauli webs. This allows us to determine, before performing any stabiliser decompositions, whether a particular error realisation results in a rejected shot. If any of the closed Pauli webs (appendix F and G) are violated (returning -1 parity), the shot is discarded and we proceed onto the next shot. If none of the closed Pauli webs are violated, only then do we perform a cutting followed by any necessary further secondary decomposition(s).

8.3 Porting into large near-Clifford circuits

For both approaches, a series of stabiliser tableaus corresponding to terms in the stabiliser decomposition is obtained at the end. With a stabiliser decomposition of χ terms,



one would have to simulate χ different numerical experiments at the escape stage of cultivation (per shot) for an end-to-end simulation. Even if one has to port those states into a much

larger Clifford circuit with thousands of qubits, the mean number of numerical experiments required is still proportional to the mean number of terms in the stabiliser decomposition. Effectively, we can also simulate any Clifford circuit applied subsequent to the cultivation circuit, such as tomography. Or even a single T-count noiseless projection on to a $\langle \bar{T}|$ with marginal ($2\times$) overhead. In an unlikely scenario where χ is very large for a particular error realisation, we can always count that particular shot as a logical failure to obtain a conservative bound on the logical error rate.

We shall move on to explicit numerical experiments implemented via optimisations around `tsim` [3] next.

9 Numerical Methods

In this section and the next (section 10), we present numerical simulation methods and results for the $d = 3$ $|T\rangle$ state cultivation circuit, based around our previously discussed cutting decomposition. The full pipeline achieves ∼4 million exact shots per second on a laptop at $p = 5 \times 10^{-4}$.

9.1 Parametric Cutting and `tsim` Integration

Since the $d = 3$ cultivation circuit contains non-Clifford gates, standard stabiliser simulation is insufficient. Instead, we evaluate stabiliser decompositions of parametric ZX-diagrams, whose scalar amplitudes depend on binary variables not fixed at compile time, enabling exact Monte Carlo sampling of detector and observable outcomes.

Unlike [1], which applies SD6 circuit-level noise throughout the entire protocol, we apply the same circuit-level noise model (with physical noise parameter p) to the injection and cultivation stages only. A noiseless projection circuit is used to calculate logical error rates at the end of the circuit. Within the noisy injection and cultivation stages, two sources of runtime uncertainty exist in the simulation. *f-parameters* represent the outcomes of depolarising noise channels and the particular Pauli error (or identity) realised is encoded as binary f-parameters sampled freshly for each shot. *m-parameters* represent unmeasured output qubits, essentially the open legs of the ZX-diagram after all detectors and observables have been plugged in, whose values must be drawn from the correct Born-rule distribution conditioned on the noise realisation. The noisy projection onto a particular detector outcome pattern is thus determined jointly by the f- and m-parameters: each shot corresponds to a specific noise realisation (f) and a sampled set of unmeasured outputs (m), and the full detector sample is read off from the resulting plugged ZX-diagram.

The ZX-diagram evaluated by the sampler is not the circuit itself but the *sampling graph* constructed by `tsim` [3] (denoted by G), which is equivalent to the circuit's ZX-representation composed with its adjoint, with data qubits traced out. Detector and observable outcomes remain as open legs (m-parameters) of this doubled graph (figure 7). Plugging each leg with a binary value $m_i \in \{0, 1\}$ closes the diagram into a scalar proportional to the Born-rule probability of that outcome pattern. For a given noise realisation $\mathbf{f}$, the sampler enumerates all $M = 2^N$ pluggings and draws the detector outcome from the resulting categorical distribution (section 9.4.2). Henceforth, G denotes this sampling graph (still a ZX-diagram) and $\tilde{G}_j$ its stabiliser decomposition terms unless stated otherwise.

Rather than treating each parameter assignment as an independent simulation, we implement a custom sampler pipeline within the `tsim` [3] framework that evaluates the parametric amplitude $\mathcal{S}(\mathbf{f}, \mathbf{m})$ for many assignments simultaneously, achieving up to ∼132k shots/s on a laptop. The pipeline is compiled end-to-end using JAX [20].

9.2 Stabiliser Rank Decomposition via Spider Cutting

The primary decomposition method is spider cutting, as described previously. The cutting decomposition expresses a parametric ZX-graph G containing non-Clifford (T gate) phases as a sum of Clifford graphs:

$$G = \sum_{i=1}^{L} \tilde{G}_i, \qquad \text{tcount}(\tilde{G}_i) = 0 \ \ \forall\, i \ , \tag{17}$$

where $\text{tcount}(\tilde{G}_i)$ is the T-count (number of spiders with phase $\in \{\pi/4, 3\pi/4, 5\pi/4, 7\pi/4\}$) in ZX-graph $\tilde{G}_i$.

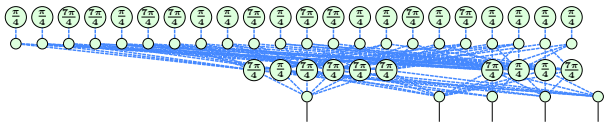
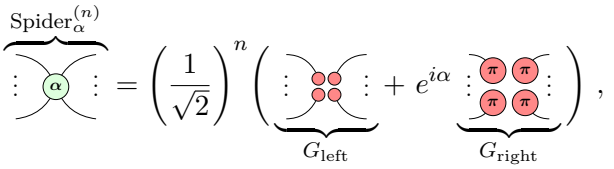


Figure 7: Sampling graph of the main component of the $d = 3$ T gate cultivation circuit, as represented in `pyzx_param`. Boundary vertices (open legs) correspond to detector and observable outcomes (m-parameters); plugging each with $m_i \in \{0, 1\}$ yields a scalar proportional to the Born-rule probability of that outcome pattern. Non-Clifford phases ($\pi/4$, $7\pi/4$) arise from the T and $T^\dagger$ gates.

9.2.1 Cutting Identity

For a Z-spider (or X-spider, reverse the colourings) with phase α and n legs connected to neighbours $\{v_1, \ldots, v_n\}$, the cutting identity reads [10]:



(18)

where G_{left} removes the spider and replaces each leg with a new opposite-colour spider of phase 0 connected to the corresponding neighbour, and G_{right} is the same but each new spider has phase π. For Hadamard edges, the created spider is same-colour (since H commutes through as a colour change).

This identity extends to the *parametric* setting: when the cut spider carries parametric phase variables $\{p_1, \ldots, p_k\}$ from noise channels, the right branch acquires an additional sign factor $(-1)^{\bigoplus_j p_j}$, tracked via the scalar's `phasevars_pi` field. This ensures that the noise dependence is faithfully preserved through the decomposition, leaving the noisy projection onto any particular error pattern exact.

9.2.2 Cutting Algorithm

Given a ZX-graph represented as a `pyzx_param` graph object (figure 7) [21], the algorithm proceeds as follows:

1. Apply `full_reduce(paramSafe=True)` to simplify (fuses spiders, removes identities, preserves parametric phases).

2. Select a non-Clifford spider to cut ($\alpha \in \{\pi/4, 3\pi/4, 5\pi/4, 7\pi/4\}$) using the *fewest-neighbours* heuristic, which minimises the $(1/\sqrt{2})^n$ scalar penalty. The spider must not be adjacent to boundary vertices.

3. Apply the cutting identity (18) $\rightarrow$ two new graphs, each with one fewer T gate.

4. Apply `full_reduce(paramSafe=True)` to each new graph (*inter-cut reduction*).

5. Recurse until all terms are Clifford (T-count $= 0$).

The inter-cut reduction at step 4 is critical: by simplifying after each cut, the algorithm exploits cancellations between spider fusions and identity removals that are invisible to methods operating on the unreduced graph.

9.2.3 Relation to BSS Decomposition

After cutting reduces the T-count as far as possible, any remaining non-Clifford terms are finished with BSS decomposition [22] (`find_stab` from `tsim`). In practice, cutting with inter-cut `full_reduce` eliminates all T gates for the cultivation circuit without needing the BSS fallback. The hybrid approach: cutting first then BSS second produces significantly fewer Clifford terms than pure BSS, because intermediate simplification exploits cancellations that BSS misses. As a reference, the native BSS only stabiliser decomposition of the sampling graph, G, in `tsim` produces 16,627 graphs (compared to 120 graphs from the cutting then BSS decomposition, see next subsection). This makes the BSS only decomposition far too impractical for any realistic computation.

Before compilation, `phasevars_pi` terms created by cutting (encoding $(-1)^{\bigoplus \mathcal{S}}$ for a set $\mathcal{S}$ of parameter variables) are converted to `phasevars_pi_pair` format: $(-1)^{(\bigoplus \mathcal{S}) \cdot 1}$, which the compiler handles as C-terms (section 9.4.1).

9.3 Sub-Component Product Factorisation

After stabiliser decomposition and `full_reduce`, the fully-plugged ZX-graph of the $d = 3$ cultivation circuit disconnects into K independent subcomponents (connected components of the ZX-

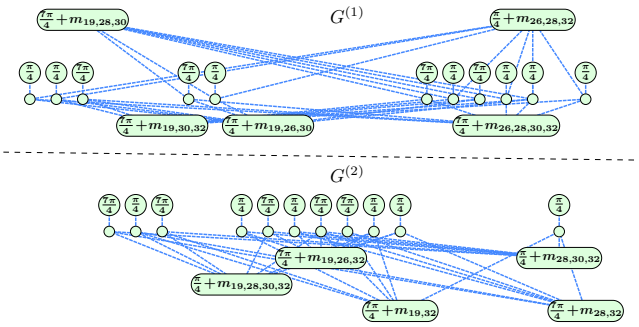


Figure 8: The fully-plugged sampling graph for the largest connected component, after `full_reduce` with parametric-safe rewriting. The graph disconnects into two sub-components $G^{(1)}$ and $G^{(2)}$, each containing 16 T gates (green nodes with $\frac{\pi}{4}$ or $\frac{7\pi}{4}$ phases). The five free outputs have been plugged with measurement-outcome parameters $m_{19}, m_{26}, m_{28}, m_{30}, m_{32}$, which appear as parametric phases on internal Z-spiders. Each sub-component is independently decomposed via spider cutting and BSS into 32 Clifford terms. Sub-component factorisation reduces the evaluation cost from $32 \times 32 = 1024$ cross-term pairs to $32 + 32 = 64$ independent term evaluations per measurement-outcome combination. All measurement-outcome parameters m_k carry an implicit factor of π, for example, a label m_{19} on a Z-spider denotes a phase of $m_{19}\pi \in \{0, \pi\}$.

graph), enabling a factored evaluation:

$$\mathcal{S}(\mathbf{f}, \mathbf{m}) = \prod_{k=1}^{K} \mathcal{S}_k(\mathbf{f}_k, \mathbf{m}_k), \tag{19}$$

where each sub-component amplitude is a sum over the G_k Clifford graphs belonging to that sub-component:

$$\mathcal{S}_k(\mathbf{f}_k, \mathbf{m}_k) = \sum_{g=1}^{G_k} s_{k,g}(\mathbf{f}_k, \mathbf{m}_k)\,. \tag{20}$$

For the $d = 3$ cultivation circuit: $K = 2$, with $G_1 = G_2 = 32$ Clifford graphs per sub-component (64 total). The remaining 56 graphs belong to 28 single-output components evaluated autoregressively, giving 120 graphs in total across the full circuit.

9.4 Compiled Scalar Graph Evaluation

9.4.1 Term Types

Each fully reduced Clifford graph g is compiled into a `CompiledScalarGraphs` structure [22], which decomposes the scalar amplitude into a product of four algebraically distinct term types. These arise naturally from the structure of a simplified ZX-diagram: after `full_reduce`, the remaining graph features such as individual spiders, phase gadgets, and parity constraints, each contribute a characteristic functional form to the scalar. The amplitude factorises as

$$s_g(\mathbf{p}) = \omega^{\phi_g} \cdot \lambda_g \cdot 2^{r_g} \cdot \prod_{t=1}^{N_A^{(g)}} A_t \cdot B \cdot C \cdot \prod_{t=1}^{N_D^{(g)}} D_t\,, \tag{21}$$

where $\omega = e^{i\pi/4}$, ϕ_g is a static phase index, $\lambda_g = a + b\omega + ci + d\bar{\omega}$ is an exact scalar factor (stored as four integer coefficients), and r_g is a power-of-two scaling exponent. All term types depend on binary row sums of the parameter vector $\mathbf{p} = [\mathbf{f}, \mathbf{m}]$: linear functions $r_t = \bigoplus_j B_{t,j}\, p_j$ over $\mathbb{F}_2$.

A-terms (node terms). Each surviving interior spider in the reduced graph contributes a factor $A_t = 1 + \omega^{\alpha_t}$, where $\alpha_t = (4\, r_t + c_t) \bmod 8$. The row sum r_t encodes how the spider's phase depends on runtime parameters, while c_t captures its fixed Clifford phase.

B-terms (half-pi terms). Phase gadgets that reduce to $\pi/4$-multiples contribute a collective phase $B = \omega^{\beta}$, where $\beta = \sum_t (r_t \cdot \tau_t) \bmod 8$ and $\tau_t \in \{0, \ldots, 7\}$ are compile-time type constants encoding each gadget's base phase.

C-terms (pi-pair terms). Parity constraints between pairs of π-phase contributions yield a collective sign $C = (-1)^{\sum_t (\psi_t \cdot \phi_t) \bmod 2}$, where ψ_t and ϕ_t are binary row sums each with an additive constant bit. In our pipeline, these arise specifically from the `phasevars_pi_pair` conversion of cutting-generated sign factors (Sec. 9.2.3).

D-terms (phase-pair terms). Pairs of phase gadgets that resist further simplification contribute $D_t = 1 + \omega^{\alpha_t} + \omega^{\beta_t} - \omega^{(\alpha_t + \beta_t) \bmod 8}$, with two independent row sums. These are the dominant cost: $N_D \approx 1408$ total across both sub-components, compared with $\sim$28 A-terms in the $d = 3$ cultivation circuit. The total amplitude is: $\mathcal{S}(\mathbf{p}) = \sum_{g=1}^{G} s_g(\mathbf{p})$.

9.4.2 Enumeration-Based Sampling

The $d = 3$ cultivation circuit's sampling graph has $N = 5$ unmeasured output bits, giving $M =$

$2^N = 32$ possible m-parameter combinations. For each noise realisation $\mathbf{f}$, we enumerate all M combinations and compute the noisy projection probability:

$$P(\mathbf{m}_c \mid \mathbf{f}) = \frac{\prod_{k=1}^{K} |\mathcal{S}_k(\mathbf{f}_k,\, \mathbf{m}_{c,k})|}{\sum_{c'=1}^{M} \prod_{k=1}^{K} |\mathcal{S}_k(\mathbf{f}_k,\, \mathbf{m}_{c',k})|} \,. \tag{22}$$

This eliminates the autoregressive sampling chain, which would require $N+1$ sequential evaluation levels, in favour of a single parallel evaluation of all M combinations at the fully-plugged level. The detector outcomes for each shot are then determined by the sampled $(\mathbf{f}, \mathbf{m})$ pair.

9.4.3 BLAS-Accelerated Binary Row Sums

All term types require binary row sums $r_{g,t} = \bigoplus_{j=1}^{P} B_{g,t,j}\, p_j$, where $B \in \{0,1\}^{G\times T\times P}$ is a compile-time binary tensor and $\mathbf{p} \in \{0,1\}^P$ are runtime parameters. For a batch of n parameter vectors, we reshape to a matrix multiply and reduce modulo 2:

$$\underbrace{(G{\cdot}T,\ P)}_{\text{binary matrix}} \times \underbrace{(P,\ n)}_{\text{param batch}} \xrightarrow{\bmod 2} (n, G, T)\,. \tag{23}$$

This activates multi-threaded BLAS (SIMD) on CPU via JAX/XLA. There are six such matrix multiplications per sub-component: two for D-term row sums (α and β), and one each for A, B, C-ψ, and C-ϕ.

9.4.4 Complex64 Lookup Table Arithmetic

The eighth roots of unity ω^k for $k = 0, \ldots, 7$ are precomputed as a length-8 `complex64` array. Term values are computed via integer index arithmetic on the row sums followed by table lookup, avoiding all trigonometric calls at runtime. The full evaluation uses `complex64` ($2 \times$ `float32`) throughout, replacing the four-component exact integer arithmetic (`ExactScalarArray`) of the reference implementation.

9.4.5 Split f/m Row Sum Decomposition

In the naïve enumeration pipeline, evaluating all $n \times M$ (shot, combo) pairs requires expanding f-parameters by a factor of M to form an $(nM,\ P)$ input matrix. We exploit the linearity of the binary row sum to avoid this expansion.

Since $\mathbf{p} = [\mathbf{f},\, \mathbf{m}]$ and $B = [B_f \mid B_m]$, the row sum splits additively:

$$\begin{aligned} r &= \Big(\sum_j B_{f,j}\, f_j + \sum_j B_{m,j}\, m_j\Big) \bmod 2 \\ &= (r_f + r_m) \bmod 2 \ . \end{aligned} \tag{24}$$

The f-contribution r_f is computed via a $(G \cdot T,\ n_f) \times (n_f,\ n)$ matrix multiply, executed once per batch (six small matmuls). The m-contribution r_m is computed via a $(G{\cdot}T,\ n_m) \times (n_m,\ M)$ matrix multiply that depends only on the $M = 32$ fixed m-parameter combinations and is precomputed once at compile time, stored as an (M, G, T) tensor.

Per-combo evaluation then reduces to the element-wise operation

$$r^{(c)} = (r_f + r_m^{(c)}) \bmod 2\,, \tag{25}$$

followed by lookup-table evaluation and product accumulation. This reduces the dominant matrix-multiply dimension from nM to n columns, a factor of $M = 32$ for the cultivation circuit. The identity $(a + b) \bmod 2 = ((a \bmod 2) + (b \bmod 2)) \bmod 2$ ensures the decomposition is exact.

After computing f-only row sums, we iterate over combos $c = 1, \ldots, M$. Each iteration combines precomputed m-row sums with f-row sums, evaluates all four term types via the LUT, computes per-graph products, and sums over graphs to obtain $\mathcal{S}_k(\mathbf{f}, \mathbf{m}_c)$ for each sub-component. The combo loop is unrolled at JIT trace time into a static XLA computation graph. The memory footprint is bounded by $(n, G, T_{\max})$ tensors per combo iteration ($\sim$367 MB at $n = 65{,}536$), rather than the infeasible $(n, M, G, T_{\max})$ that full materialisation would require ($\sim$12 GB).

9.4.6 Noiseless Caching

Under circuit-level, uniform depolarising noise at strength p, a fraction η of shots have all-zero f-parameters. For these noiseless shots, the noisy projection reduces to the noise-free case and the sampling distribution $P(\mathbf{m} \mid \mathbf{f} = \mathbf{0})$ is identical across all such shots, so it can be precomputed once. The sampler operates in two phases:

1. **Phase 1.** Sample noise channel outcomes. Noiseless shots ($\mathbf{f} = \mathbf{0}$) are served from the

Table 1: Sampling throughput for the $d = 3$ cultivation circuit ($N = 5$, $M = 32$, $K = 2$, batch size 65,536) on Apple M4 Pro series CPU at $p = 0.001$ with 2^{26} shots. All methods are exact.

Optimisation	**shots/s**	**Speedup**	**Key change**
Autoregressive baseline	∼20k	1.0×	Sequential $N+1$ evaluation levels
+ Enumeration + BLAS matmul	∼38k	1.9×	Single-level eval of all $M = 32$ combos
+ Complex64 LUT	∼42k	2.1×	Float32 lookup replaces exact integer arith.
+ Full-program JIT + noiseless cache	∼54k	2.7×	No Python overhead; cache ∼64% noiseless shots
+ Split f/m decomposition	∼94k	4.7×	32× matmul reduction via row-sum linearity

cached distribution via a single categorical draw. Noisy shots are collected into a separate buffer.

2. **Phase 2.** All noisy shots are consolidated and processed through the full split f/m pipeline in fixed-size batches (avoiding JIT recompilation).

At $p = 0.001$, approximately $\eta \approx 63.9\%$ of shots are noiseless, so only ∼36% require full amplitude evaluation.

9.4.7 Full-Program JIT Compilation

The entire sampling function, including

1. noise channel categorical draws,
2. per-component enumeration,
3. per-sub-component split f/m evaluation, and
4. categorical output sampling,

is compiled into a single `jax.jit`-compiled function. All Python control flow (loops over components, sub-components, and combos) is resolved at trace time, producing a static XLA computation graph with no Python dispatch overhead at runtime.

9.5 Performance

The split f/m sampler achieves 94.8 thousand shots/s with exact sampling at $p = 0.001$. All Monte Carlo estimates are computed with $2^{26} \approx 67$ million shots (except at $p = 0.0005$, which uses 2^{28} shots). Every shot evaluates the exact non-Clifford amplitude of the T gate circuit via a noiseless projection, a departure from [1], which replaces $T(T^\dagger) \to S(S^\dagger)$ to enable Clifford-only simulation via `stim` [13]. See figure 10 for a shots per second comparison. Our approach requires no

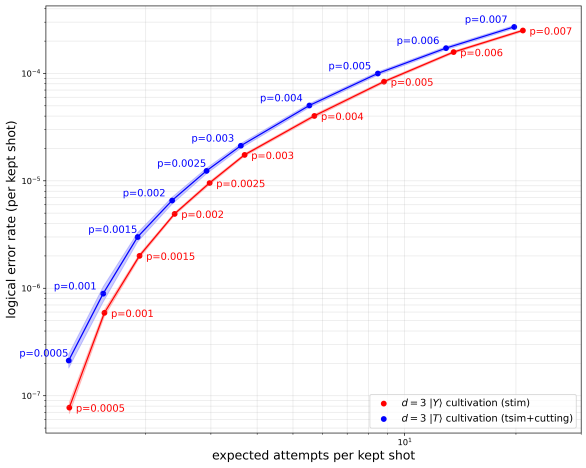


Figure 9: Logical error rate (LER) of the $d = 3$ T gate cultivation circuit as a function of physical noise strength p, sampled with 2^{26} shots per data point via exact noiseless projection (in blue). For comparison, the S gate proxy of the equivalent noiseless projection circuit simulated with `stim` is shown in red. The T gate LER is approximately 2× higher than the S gate proxy across the plotted noise range, consistent with the proxy underestimating detector sensitivity by a factor of two as noted in figure 13 of [1].

such gate substitution, the stabiliser decomposition (and BSS fall back per term) handles the full non-Clifford circuit directly.

At $p = 0.001$, the full run produces a post-selection rate of 0.651, with 39 errors in 43.7 million post-selected shots, corresponding to logical error rate of $\approx 8.9 \times 10^{-7}$. This is, to our knowledge, the first exact (non-Clifford-proxy) Monte Carlo estimate of the logical error rate for the $d = 3$ T gate cultivation circuit at operationally relevant noise strengths, made feasible by the combination of cutting decomposition and the optimised sampling pipeline. At $p = 0.0005$, the sampler is only ∼34× slower (∼132,400 shots a second) than the fully Clifford proxy simulation via `stim` [13] using S gates (∼ 4.47 million shots a second). Figure 9 shows the logical error rate as a function of physical noise strength for

the exact T gate simulation (with noiseless projection) alongside its equivalent S gate Clifford proxy. Across the operationally relevant regime $p \in [10^{-4}, 10^{-3}]$, the T gate logical error rate tracks approximately within a factor of two relative to the S gate proxy LER value. Full pseudocode for this decomposition, compilation, and sampling pipeline is provided in Appendix A.

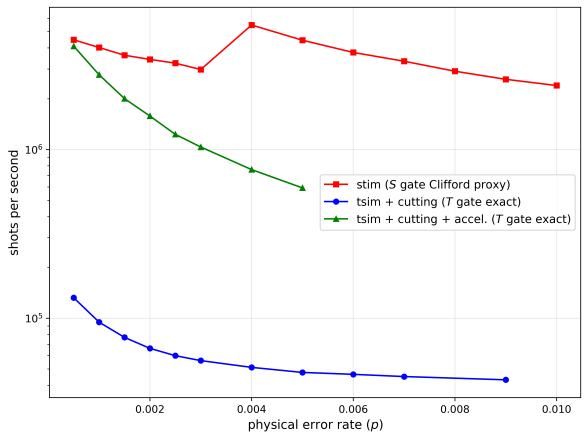


Figure 10: Sampling throughput (shots per second) as a function of physical circuit-level noise strength p for the $d = 3$ cultivation circuit. The S gate Clifford proxy via `stim` (red) achieves $\sim$4.5 million shots/s. The baseline exact T gate simulation via `tsim`+cutting (blue) reaches $\sim$132,000 shots/s at $p = 5 \times 10^{-4}$. The accelerated pipeline (green), combining sparse geometric channel sampling, cross-batch deduplication, and scan-based evaluation, achieves $\sim$4.1 million shots/s at $p = 5 \times 10^{-4}$. This is a $\sim 31\times$ speed-up over the baseline and only $\sim 1.1\times$ slower than `stim`. The accelerated sampler's advantage diminishes as the dedup compression ratio falls with increasing p.

We shall now move onto an even faster implementation of the $d = 3$ circuits, reaching $\sim$4 million shots per second on a laptop at $p = 0.0005$.

10 Accelerated Sampling Pipeline

Several further optimisations target the remaining bottlenecks in the sampling pipeline identified by profiling: random number generation in the channel sampler (Phase 1), redundant tensor network evaluations across shots sharing identical fault patterns (Phase 2), and JIT compilation overhead in the combo evaluation loop. Combined, these yield a $\sim 30\times$ additional throughput improvement over the split f/m baseline of section 9.4.5 at $p = 10^{-3}$, while preserving exact sampling.

10.1 Sparse Geometric Channel Sampling

The JIT-compiled channel sampler of section 9.4.7 uses the Gumbel-Max trick [23, 24]: for each channel with K outcomes, it draws K i.i.d. Gumbel random variables, adds them to the log-probabilities, and returns the argmax. For the d=3 cultivation circuit with $N_{\text{ch}} = 124$ channels averaging $\bar{K} \approx 4$ outcomes each, this generates $N_{\text{ch}} \cdot \bar{K} \cdot B \approx 10^9$ random numbers per batch of $B = 2{,}097{,}152$ shots. An intermediate step replaces the Gumbel-Max sampler with inverse CDF sampling via `searchsorted`, reducing random number generation by a factor of $\bar{K}$ to $N_{\text{ch}} \cdot B$ draws per batch. However, at low noise, even one uniform per channel per shot is wasteful: the identity outcome dominates with probability $P(\text{identity}) \approx 1 - 3p \approx 0.9985$ at $p = 5 \times 10^{-4}$, so the vast majority of draws produce no effect.

We exploit this sparsity via geometric skip sampling: instead of asking "what happened at each shot?" for every channel, we ask "which shot fires next?" The geometric distribution $\text{Geom}(p_{\text{fire}})$ gives the gap between consecutive non-identity ("fire") events, where $p_{\text{fire}} = 1 - P(\text{identity})$ is precomputed per channel. Fire positions are obtained by cumulative summation of geometric draws, giving sorted (and therefore cache-friendly) write positions. For channels with multiple non-identity outcomes, a single conditional uniform per fire event selects among them via `searchsorted` on a precomputed conditional CDF. The selected outcome's XOR pattern is applied in-place to the result array.

The total random number budget is $\sim 2 N_{\text{ch}} B \bar{p}_{\text{fire}}$: one geometric draw per skip plus one conditional uniform per fire event. At $p = 5 \times 10^{-4}$, this amounts to $\sim$500K draws instead of $\sim$260M (inverse CDF) or $\sim 10^9$ (Gumbel-Max) per batch. The entire sampler runs in NumPy, eliminating JAX dispatch overhead for the $N_{\text{ch}} = 124$ per-channel operations.

10.2 Unique-Pattern Deduplication with Persistent Cache

At low physical error rates, the channel sampler produces highly repetitive fault patterns: among the $\sim$36% of shots that are noisy at $p = 0.001$, the vast majority share one of a small number of distinct f-parameter vectors (those with exactly one

or two channels in a non-identity state). Since the enumerated sub-component evaluation of section 9.4.2 depends on $\mathbf{f}$ only through the $n_f = 37$ f-parameters selected by the enum component, we can eliminate this redundancy by evaluating only the $U \ll B$ unique f-patterns and scattering the results back.

The deduplication proceeds in three steps:

1. **Hash.** Each shot's n_f-bit f-pattern is packed into a single `int64` scalar via the inner product $h_i = \mathbf{f}_i \cdot (2^0, 2^1, \ldots, 2^{n_f-1})$, computed as a BLAS matrix-vector product. This is exact for $n_f \leq 63$.

2. **Unique.** `numpy.unique` on the scalar hashes returns U unique values and an inverse index σ mapping each shot to its unique pattern. Operating on `int64` scalars, this runs in $O(B \log B)$ time, which is vastly faster than row-wise unique on (B, n_f) boolean arrays.

3. **Decode.** The U unique hashes are decoded back to n_f-bit patterns via bitwise extraction, padded to the next power of two for JIT shape stability, and passed to the split f/m evaluator.

The per-sub-component magnitudes $|\mathcal{S}_k(\mathbf{f}_u, \mathbf{m}_c)|$ are evaluated for each unique pattern $u = 1, \ldots, U$ and each combo $c = 1, \ldots, M$. The joint magnitudes are then scattered back to all B shots via the inverse index: $P_i = \prod_k |\mathcal{S}_k|[\sigma(i)]$, after which each shot draws independently from its own categorical distribution.

The deduplication is performed in NumPy **outside** the JIT boundary, avoiding JAX's 64-bit integer limitations, while the tensor network evaluation remains inside a JIT-compiled function operating on the (much smaller) unique pattern array. At $p = 5 \times 10^{-4}$, within-batch deduplication reduces the number of split f/m evaluations per batch from ∼400,000 noisy shots to ∼500 unique patterns—an 800× reduction.

Cross-batch persistent cache. The within-batch dedup finds $U \ll B$ unique patterns per batch, but many patterns recur across batches, especially at low noise where most noisy shots have only one or two channels firing. A persistent dictionary mapping `int64` hashes to magnitude vectors is maintained across all batches. After dedup, each unique hash is checked against this cache; only genuinely new patterns trigger the split f/m evaluation. At $p = 5 \times 10^{-4}$, approximately $U \approx 500$ unique patterns appear per batch, but the total number of distinct patterns across all $n_{\text{batches}} = 512$ batches is only ∼1,000. The persistent cache therefore reduces total evaluations from $U \times n_{\text{batches}} \approx 256{,}000$ to ∼1,000—a further ∼250× reduction beyond within-batch dedup alone.

10.3 Scan-Based Combo Evaluation

The split f/m evaluator of section 9.4.5 iterates over the $M = 32$ measurement-outcome combos in a Python `for`-loop that is unrolled at JIT trace time into M copies of the loop body. This increases compilation time and instruction-cache pressure. We replace this with `jax.lax.scan`, which compiles the loop body once and iterates at runtime. Additionally, the D-term values (section 9.4.1) are precomputed into a 4-entry lookup table per (G, T_D) position, indexed by the two parity bits of the f- and m-row sums, eliminating redundant complex-exponential computation inside the loop body. Together, these changes halve JIT warmup time and reduce per-batch evaluation cost.

10.4 Results

Table 3 (and figure 11) reports throughput and logical error rates for the combined pipeline at 2^{29} to 2^{32} shots per noise strength on Apple M4 Pro CPU with batch size $B = 2{,}097{,}152$. At $p = 5 \times 10^{-4}$, the accelerated pipeline achieves ∼4.1 million shots/s, within ∼1.1× of the Clifford-proxy `stim` throughput. The `stim` reference throughput is measured at 2^{29} shots in a single call; unlike the exact sampler, which loops over fixed 2M-shot batches, `stim` allocates all shots at once and its throughput is sensitive to call size. We did not explore batching strategies for `stim` that could improve its throughput. At $p = 10^{-3}$, this represents a ∼30× improvement over the split f/m baseline (see table 2). All optimisations preserve exact sampling: the output distribution is identical to the baseline pipeline. The logical error rate scales as $\sim p^3$, consistent with a distance-3 code. Pseudocode for the full accelerated pipeline is provided in Appendix B. The pipeline's throughput degrades at higher physical error rates be-

Table 2: Accelerated sampling optimisations for the $d = 3$ cultivation circuit ($N = 5$, $M = 32$, $K = 2$, batch size 2,097,152) on Apple M4 Pro CPU at $p = 10^{-3}$ with 2^{29} shots. Speed-ups are relative to the split f/m baseline of table 1. All methods are exact.

Optimisation	**shots/s**	**Speedup**	**Key change**
Split f/m baseline (table 1)	∼94k	1.0×	Batch size 65,536
+ Inverse CDF + deduplication	∼974k	10.4×	$\bar{K}\times$ fewer draws; eval only $U \ll B$ unique f-patterns
+ Geometric skip + persistent cache + scan	∼2,778k	∼30×	Sparse fire events; cross-batch cache; single-body scan

cause the persistent cache's compression ratio decreases and the sparse channel sampler generates more fire events, increasing the fraction of shots that require full ZX/tensor-network evaluation. In addition, post-selection rates are consistent

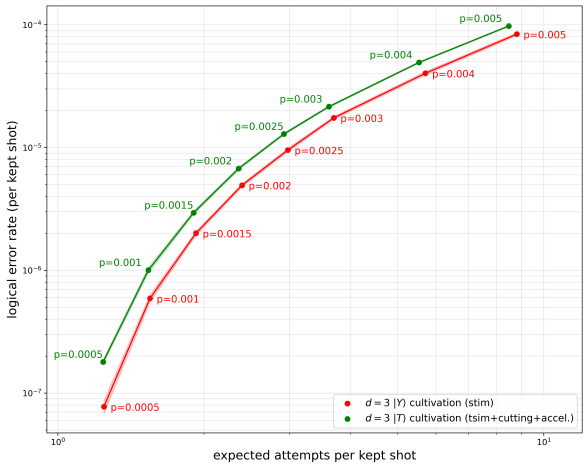


Figure 11: Logical error rate versus expected attempts per kept shot for the accelerated $|T\rangle$ simulation (green, 2^{29} to 2^{32} shots) and the $|Y\rangle$ (S gate) Clifford proxy via `stim` (red). Same format as figure 9, shaded bands show binomial standard error.

across all three configurations and decrease with increasing physical error rate, as shown in figure 12 as discard ratios (1−PSR).

11 Summary

Magic state cultivation circuits with moderate T-count (in the $\sim$ 50 range) can be simulated exactly via stabiliser decomposition with manageable overhead per shot relative to pure stabiliser circuits. We provide an optimised numerical pipeline based on `tsim`, incorporating spider cutting, BLAS-accelerated evaluation, enumeration-based sampling, and an accelerated two-phase architecture combining sparse geometric channel sampling, persistent cross-batch deduplication, and scan-based evaluation (section 10), achieving ∼4 million exact shots per second for the d=3 circuit at $p = 5 \times 10^{-4}$. This is within a factor of ∼1.1 of the fully Clifford proxy via `stim`. We note that our pipeline is compiled and optimised for the specific d=3 cultivation circuit, whereas `stim` is a general-purpose Clifford simulator. Extending these simulations to the $d = 5$ cultivation circuit, potentially with GPU-accelerated `tsim`, is left for future work.

We note that several concurrent works on magic state cultivation have recently appeared [25–27] or been updated [28] on the arXiv. It would be interesting to investigate whether our simulation method extends to other cultivation circuit variants [29]. We note that the throughput in our numerical experiments is several orders of magnitude higher than that of the competing approach [27] and out-performs [30] in terms of average shots per second by a factor of 11.7, without the aid of a pricey H800 Nvidia GPU[9].

See also Appendix D for a potential approach to further reducing the total number of stabiliser terms required for simulating the double-checking circuits.

12 Acknowledgements

We want to thank K.H.W's wife for all the numerical support in simulating the magic state cultivation circuits exactly. None of this would be possible without her support. We also acknowledge useful feedback from Yves Vollmeier, Austin Fowler, Isaac Kim, Beatriz Dias, Rafael Haenel and our journal submission editors/reviewers. Z.Z was supported by the ERC Consolidator Grant # 864828 "Algebraic Foundations of Supersymmetric Quantum Field Theory" (SCFTAlg). Packages such as `zxlive`, `pyzx` [12], `tsim` [3], magic state cultivation simulation python code [19] and cutting stabiliser decomposition python

[9] Googled price of Nvidia H800 as of Feb 2026, ∼$30,000 (USD). The Macbook Pro used for our computation cost ∼$2,000 (USD).

Table 3: Accelerated sampling results for the d=3 cultivation circuit (batch size 2,097,152, Apple M4 Pro CPU). All methods are exact (non-Clifford).

p	Shots	shots/s	PSR	Errors	n_{kept}	LER
5×10^{-4}	2^{32}	4,102,367	0.807	622	3,465,069,335	1.8×10^{-7}
1×10^{-3}	2^{29}	2,777,295	0.651	351	349,529,061	1.0×10^{-6}
1.5×10^{-3}	2^{29}	2,005,014	0.525	831	282,110,746	2.9×10^{-6}
2×10^{-3}	2^{29}	1,580,499	0.424	1,540	227,748,263	6.8×10^{-6}
2.5×10^{-3}	2^{29}	1,230,763	0.343	2,373	183,908,903	1.3×10^{-5}
3×10^{-3}	2^{29}	1,035,449	0.277	3,201	148,548,040	2.2×10^{-5}
4×10^{-3}	2^{29}	760,769	0.181	4,785	96,970,177	4.9×10^{-5}
5×10^{-3}	2^{29}	592,605	0.118	6,193	63,357,051	9.8×10^{-5}

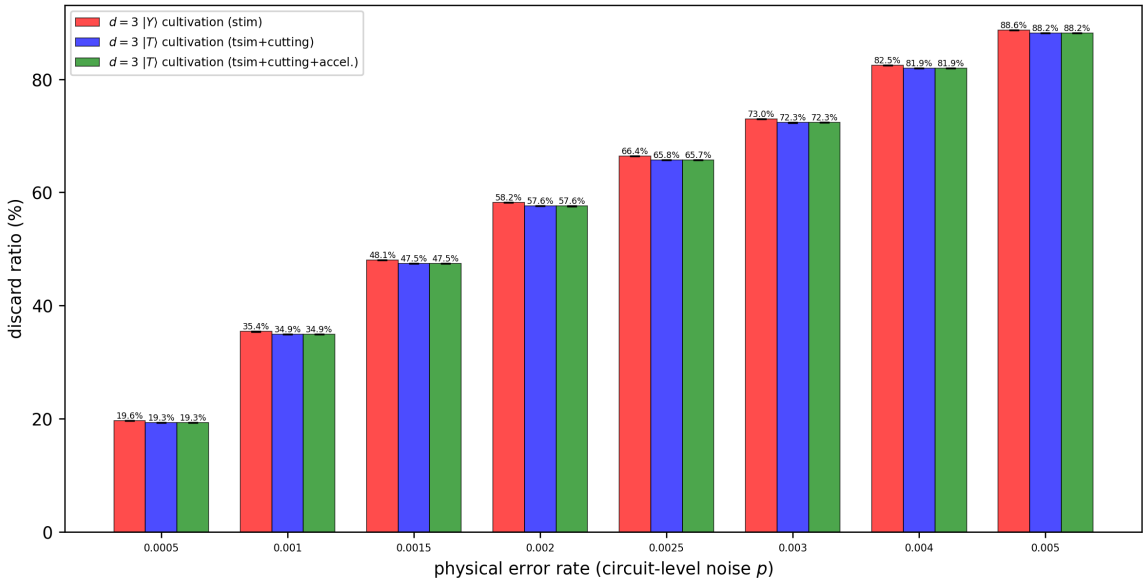


Figure 12: Discard ratio (1−PSR) across physical error rates $p \in \{0.0005, \ldots, 0.005\}$ for three configurations: stabiliser S gate proxy via `stim` (red), unaccelerated T gate via `tsim` (blue), and accelerated T gate via `tsim` with the optimisations of section 10 (green). All three circuits share the same noise model and post-selection criteria. The S and T gate PSR values are nearly indistinguishable, confirming that the non-Clifford gate does not affect the discard rate. Each bar is generated from at least 2^{26} shots.

code [9] have been extensively used in the preparation of this manuscript. All of the numerics were performed on an Apple M4 Macbook Pro, or devices of less computational power. Minimally reproducible code available at [31]. The wording in certain sections of this manuscript had been refined using LLMs.

References

[1] Craig Gidney, Noah Shutty, and Cody Jones. Magic state cultivation: growing T states as cheap as CNOT gates, 2024. URL https://arxiv.org/abs/2409.17595.

[2] Kwok Ho Wan and Zhenghao Zhong. Cutting stabiliser decompositions of magic state cultivation with ZX-calculus, 2025. URL https://arxiv.org/abs/2509.01224.

[3] Rafael Haenel et al. tsim: Universal Quantum Circuit Sampler based on ZX Stabilizer Rank Decomposition, 2026. URL https://github.com/QuEraComputing/tsim. Version 0.1.0, Apache-2.0 License.

[4] Hammam Qassim, Hakop Pashayan, and David Gosset. Improved upper bounds on the stabilizer rank of magic states. Quantum, 5:606, December 2021. ISSN 2521-327X. DOI: 10.22331/q-2021-12-20-606. URL http://dx.doi.org/10.22331/q-2021-12-20-606.

[5] Julien Codsi. Cutting-Edge Graphical Stabiliser Decompositions for Classical Simula-

tion of Quantum Circuits. Master's thesis, University of Oxford, 2022.

[6] Aleks Kissinger, John van de Wetering, and Renaud Vilmart. Classical simulation of quantum circuits with partial and graphical stabiliser decompositions. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. DOI: 10.4230/LIPICS.TQC.2022.5. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TQC.2022.5.

[7] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. Physical Review A, 71(2), February 2005. ISSN 1094-1622. DOI: 10.1103/physreva.71.022316. URL http://dx.doi.org/10.1103/PhysRevA.71.022316.

[8] Daniel Litinski. Magic State Distillation: Not as Costly as You Think. Quantum, 3:205, December 2019. ISSN 2521-327X. DOI: 10.22331/q-2019-12-02-205. URL http://dx.doi.org/10.22331/q-2019-12-02-205.

[9] Matthew Sutcliffe and Aleks Kissinger. Procedurally Optimised ZX-Diagram Cutting for Efficient T-Decomposition in Classical Simulation. Electronic Proceedings in Theoretical Computer Science, 406:63–78, August 2024. ISSN 2075-2180. DOI: 10.4204/eptcs.406.3. URL http://dx.doi.org/10.4204/EPTCS.406.3.

[10] Matthew Sutcliffe. Novel Methods for Classical Simulation of Quantum Circuits via ZX-Calculus. PhD thesis, University of Oxford, 2025.

[11] Hector Bombin, Daniel Litinski, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. Unifying flavors of fault tolerance with the ZX calculus. Quantum, 8:1379, June 2024. ISSN 2521-327X. DOI: 10.22331/q-2024-06-18-1379. URL http://dx.doi.org/10.22331/q-2024-06-18-1379.

[12] Aleks Kissinger and John van de Wetering. PyZX: Large Scale Automated Diagrammatic Reasoning. In Bob Coecke and Matthew Leifer, editors, Proceedings 16th International Conference on Quantum Physics and Logic, Chapman University, Orange, CA, USA., 10-14 June 2019, volume 318 of Electronic Proceedings in Theoretical Computer Science, pages 229–241. Open Publishing Association, 2020. DOI: 10.4204/EPTCS.318.14.

[13] Craig Gidney. Stim: a fast stabilizer circuit simulator. Quantum, 5:497, July 2021. ISSN 2521-327X. DOI: 10.22331/q-2021-07-06-497. URL http://dx.doi.org/10.22331/q-2021-07-06-497.

[14] Benjamin Rodatz, Boldizsár Poór, and Aleks Kissinger. Fault Tolerance by Construction, 2025. URL https://arxiv.org/abs/2506.17181.

[15] Pedro Sales Rodriguez, John M. Robinson, Paul Niklas Jepsen, Zhiyang He, Casey Duckering, Chen Zhao, Kai-Hsin Wu, Joseph Campo, Kevin Bagnall, Minho Kwon, Thomas Karolyshyn, Phillip Weinberg, Madelyn Cain, Simon J. Evered, et al. Experimental Demonstration of Logical Magic State Distillation, 2024. URL https://arxiv.org/abs/2412.15165.

[16] Sergey Bravyi, Graeme Smith, and John A. Smolin. Trading Classical and Quantum Computational Resources. Phys. Rev. X, 6:021043, Jun 2016. DOI: 10.1103/PhysRevX.6.021043. URL https://link.aps.org/doi/10.1103/PhysRevX.6.021043.

[17] Mark Koch, Richie Yeung, and Quanlong Wang. Speedy Contraction of ZX Diagrams with Triangles via Stabiliser Decompositions, 2023. URL https://arxiv.org/abs/2307.01803.

[18] Yves Vollmeier. Graphical Stabilizer Decompositions for Multi-Control Toffoli Gate Dense Quantum Circuits, 2025. URL https://arxiv.org/abs/2503.03798.

[19] Craig Gidney. Data for "Magic state cultivation: growing T states as cheap as CNOT gates". Zenodo, September 2024. DOI: 10.5281/zenodo.13777072.

[20] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

[21] Rafael Haenel, Matthew Sutcliffe, Aleks Kissinger, John van de Wetering, et al. PyZX-Param: PyZX with extensions for

parameterized Pauli vertices, 2026. URL https://github.com/rafaelha/pyzx. Fork of PyZX based on ParamZX [10].

[22] Matthew Sutcliffe. Smarter k-Partitioning of ZX-Diagrams for Improved Quantum Circuit Simulation, 2024. URL https://arxiv.org/abs/2409.00828.

[23] Chris J. Maddison, Daniel Tarlow, and Tom Minka. A Sampling, 2015. URL https://arxiv.org/abs/1411.0030.

[24] E. J. Gumbel and J. Lieblein. Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures. US Government Printing Office, Washington, DC, 1954.

[25] Kaavya Sahay, Pei-Kai Tsai, Kathleen Chang, Qile Su, Thomas B. Smith, Shraddha Singh, and Shruti Puri. Fold-transversal surface code cultivation, 2025. URL https://arxiv.org/abs/2509.05212.

[26] Jahan Claes. Cultivating T states on the surface code with only two-qubit gates, 2025. URL https://arxiv.org/abs/2509.05232.

[27] Samyak Surti, Lucas Daguerre, and Isaac H. Kim. Efficient simulation of logical magic state preparation protocols, 2025. URL https://arxiv.org/abs/2512.23799.

[28] Yotam Vaknin, Shoham Jacoby, Arne Grimsmo, and Alex Retzker. Efficient Magic State Cultivation on the Surface Code, 2025. URL https://arxiv.org/abs/2502.01743.

[29] Zi-Han Chen, Ming-Cheng Chen, Chao-Yang Lu, and Jian-Wei Pan. Efficient Magic State Cultivation on $\mathbb{RP}^2$, 2025. URL https://arxiv.org/abs/2503.18657.

[30] Riling Li, Keli Zheng, Yiming Zhang, Huazhe Lou, Shenggang Ying, Ke Liu, and Xiaoming Sun. Soft: a high-performance simulator for universal fault-tolerant quantum circuits, 2025. URL https://arxiv.org/abs/2512.23037.

[31] Kwok Ho Wan, Zhenghao Zhong, and Ainhoa . Accelerated exact sampling for magic state cultivation, 2026. URL https://github.com/kh428/accel-cutting-magic-state. Apache-2.0 License.

[32] Aleks Kissinger and John van de Wetering. Picturing Quantum Software: An Introduction to the ZX-Calculus and Quantum Compilation.

A Sampling Pipeline Pseudocode

We present the full sampling pipeline in three stages: stabiliser decomposition via cutting (Algorithm 1), compilation and precomputation (Algorithm 2), and per-batch sampling (Algorithm 3). Throughout, $\omega = e^{i\pi/4}$, $\oplus$ denotes XOR, and $M = 2^N$ is the number of m-parameter combinations.

Algorithm 1 Cutting-based stabiliser decomposition.

Input: ZX-graph G with $\text{tcount}(G) > 0$, max iterations $I_{\max}$
Output: Set of Clifford ZX-graphs $\mathcal{T}_{\text{Cliff}}$

function DECOMPOSE($G, I_{\max}$)
  $G \leftarrow$ `full_reduce`(G, `paramSafe` = TRUE)
  $\mathcal{T} \leftarrow \{G\}$; $\mathcal{T}_{\text{Cliff}} \leftarrow \emptyset$
  for $i = 1$ **To** $I_{\max}$ **do**
    $\mathcal{T}_{\text{new}} \leftarrow \emptyset$
    for each $T \in \mathcal{T}$ **do**
      $T \leftarrow$ `full_reduce`(T, `paramSafe` = TRUE)
      if $\text{scalar}(T) = 0$ **then continue**
      end if
      if $\text{tcount}(T) = 0$ **then** $\mathcal{T}_{\text{Cliff}} \leftarrow \mathcal{T}_{\text{Cliff}} \cup \{T\}$; **continue**
      end if
      $v^* \leftarrow \arg\min_{v \in \text{cuttable}(T)} |\text{neighbours}(v)|$ ▷ fewest-neighbours heuristic
      if $v^* =$ NONE **then** ▷ BSS fallback
        $\mathcal{T}_{\text{Cliff}} \leftarrow \mathcal{T}_{\text{Cliff}} \cup \text{BSS}(T)$; **continue**
      end if
      $(G_L, G_R) \leftarrow$ CUTSPIDER(T, v^*) ▷ Eq. (18)
      $\mathcal{T}_{\text{new}} \leftarrow \mathcal{T}_{\text{new}} \cup \{G_L, G_R\}$
    end for
    $\mathcal{T} \leftarrow \mathcal{T}_{\text{new}}$
    if $\mathcal{T} = \emptyset$ **then break**
    end if
  end for
  for each $T \in \mathcal{T}$ with $\text{tcount}(T) > 0$ **do** ▷ BSS cleanup
    $\mathcal{T}_{\text{Cliff}} \leftarrow \mathcal{T}_{\text{Cliff}} \cup \text{BSS}(T)$
  end for
  Convert all `phasevars_pi` $\rightarrow$ `phasevars_pi_pair` ▷ $(-1)^{\bigoplus S} \rightarrow (-1)^{(\bigoplus S)\cdot 1}$
  return $\mathcal{T}_{\text{Cliff}}$
end function

Subroutine CUTSPIDER(G, v): spider v has phase α, n legs, parametric variables $\{p_1, \ldots, p_k\}$.
G_L: remove v; attach n new opposite-colour phase-0 spiders; scalar $\times(1/\sqrt{2})^n$.
G_R: remove v; attach n new opposite-colour phase-π spiders; scalar $\times(1/\sqrt{2})^n \cdot e^{i\pi\alpha} \cdot (-1)^{\bigoplus_j p_j}$.
For Hadamard edges: created spider is same-colour.

Algorithm 2 Compile circuit and precompute split f/m row sums.

Input: Noisy circuit $\mathcal{C}$, noise strength p, max iterations $I_{\max}$
Output: Compiled sampling program (JIT-compiled)

function COMPILE($\mathcal{C}$, p, $I_{\max}$)
  $G_{\mathrm{ZX}} \leftarrow$ `tsim.circuit_to_zx`($\mathcal{C}$)
  $\{\mathcal{K}_1, \ldots\} \leftarrow$ connected_components(G_{ZX})
  for each component $\mathcal{K}$ with N output bits **do**
    $M \leftarrow 2^N$
    $G_{\mathrm{plug}} \leftarrow$ `plug_all_outputs`($\mathcal{K}$)
    `full_reduce`(G_{plug}, `paramSafe` = TRUE)
    $\{G^{(1)}, \ldots, G^{(K)}\} \leftarrow$ connected_components(G_{plug}) ▷ K sub-components

    for $k = 1$ **To** K **do** ▷ **Decompose and compile**
      $\mathcal{T}_k \leftarrow$ DECOMPOSE($G^{(k)}$, $I_{\max}$) ▷ Alg. 1
      $\mathrm{CSG}_k \leftarrow$ `compile_scalar_graphs`($\mathcal{T}_k$) ▷ G_k graphs, A/B/C/D terms
    end for

    $\mathbf{m}_{\mathrm{combos}} \leftarrow \{0,1\}^{M \times N}$ ▷ **Enumerate** all M output combos

    for $k = 1$ **To** K **do** ▷ **Precompute m-contribution row sums**
      for $\tau \in \{D_\alpha, D_\beta, A, B, C_\psi, C_\phi\}$ **do**
        $B_m^{(\tau)} \leftarrow \mathrm{CSG}_k.\mathrm{param\_bits}_\tau[:, :, \mathrm{m\_cols}]$ ▷ $(G_k, T_\tau, n_{m,k})$
        $\mathbf{v}_m \leftarrow \mathbf{m}_{\mathrm{combos}}[:, \mathrm{m\_indices}_k]$ ▷ $(M, n_{m,k})$
        $R_m^{(\tau,k)} \leftarrow \left(\mathrm{reshape}(B_m^{(\tau)}) \times \mathbf{v}_m^\top\right) \bmod 2$ ▷ $(G_k{\cdot}T_\tau, M) \to (M, G_k, T_\tau)$
      end for
    end for

    $P(\mathbf{m} \mid \mathbf{f}=\mathbf{0}) \leftarrow$ evaluate all M combos at $\mathbf{f} = \mathbf{0}$ ▷ **Noiseless cache**
  end for
  return `jax.jit`(SAMPLEBATCH) ▷ **JIT-compile** full sampling function
end function

Algorithm 3 Per-batch sampling with split f/m evaluation.

Input: Compiled program, batch size n, PRNG key
Output: Sampled detector outcomes for n shots

function SAMPLEBATCH(program, n, key)

▷ **Phase 1: Channel sampling + noiseless cache**

$\mathbf{F} \leftarrow$ `channel_sample`(n) ▷ (n, n_f) binary matrix from depolarising noise

$\mathcal{I}_0 \leftarrow \{i : \mathbf{F}[i,:] = \mathbf{0}\}$ ▷ noiseless shots ($\sim$63.9% at $p = 0.001$)

results$[\mathcal{I}_0] \leftarrow$ `categorical`$(P(\mathbf{m} \mid \mathbf{f} = \mathbf{0}))$ ▷ single cached draw

$\mathbf{F}' \leftarrow \mathbf{F}[\overline{\mathcal{I}_0}, :]$ ▷ (n', n_f) noisy shots, $n' \approx 0.36\,n$

▷ **Phase 2: Full evaluation for noisy shots**

for each component with sub-components $k = 1, \ldots, K$ **do**

$W \leftarrow \mathbf{1}_{n' \times M}$ ▷ joint magnitudes, initialised to 1

for $k = 1$ **To** K **do**

▷ **Step A: f-only row sums** (6 matmuls, computed once)

for $\tau \in \{D_\alpha,\ D_\beta,\ A,\ B,\ C_\psi,\ C_\phi\}$ **do**

$B_f^{(\tau)} \leftarrow \mathrm{CSG}_k.\mathrm{param\_bits}_\tau[:, :, \mathrm{f\_cols}_k]$ ▷ $(G_k, T_\tau, n_{f,k})$

$R_f^{(\tau)} \leftarrow \left(\mathrm{reshape}(B_f^{(\tau)}) \ \times\ \mathbf{F}_k'^{\top}\right) \bmod 2$ ▷ $(G_k{\cdot}T_\tau,\ n') \to (n',\ G_k,\ T_\tau)$

end for

▷ **Step B: Loop over combos** (unrolled at JIT trace time)

for $c = 1$ **To** M **do**

for $\tau \in \{D_\alpha,\ D_\beta,\ A,\ B,\ C_\psi,\ C_\phi\}$ **do**

$R^{(\tau)} \leftarrow (R_f^{(\tau)} + R_m^{(\tau,k)}[c]) \bmod 2$ ▷ element-wise, (n', G_k, T_τ)

end for

▷ D-terms: LUT $\to$ product over t

$\boldsymbol{\alpha} \leftarrow (4\,R^{(D_\alpha)} + \mathbf{c}_{D_\alpha}) \bmod 8; \quad \boldsymbol{\beta} \leftarrow (4\,R^{(D_\beta)} + \mathbf{c}_{D_\beta}) \bmod 8$

$P_D \leftarrow \prod_t \left(1 + \omega[\boldsymbol{\alpha}_t] + \omega[\boldsymbol{\beta}_t] - \omega[(\boldsymbol{\alpha}_t + \boldsymbol{\beta}_t) \bmod 8]\right)$ ▷ (n', G_k)

▷ A-terms: LUT $\to$ product over t

$P_A \leftarrow \prod_t \left(1 + \omega[(4\,R_t^{(A)} + \mathbf{c}_A) \bmod 8]\right)$ ▷ (n', G_k)

▷ B-term: collective phase; C-term: collective sign

$V_B \leftarrow \omega\left[\sum_t (R_t^{(B)} \cdot \boldsymbol{\tau}_t) \bmod 8\right]; \quad V_C \leftarrow (-1)^{\sum_t ((R_t^{(C_\psi)} + c_{\psi,t}) \cdot (R_t^{(C_\phi)} + c_{\phi,t})) \bmod 2}$

▷ Sum over graphs

$S_k^{(c)} \leftarrow \sum_{g=1}^{G_k} \omega^{\phi_g} \cdot \lambda_g \cdot 2^{r_g} \cdot P_A[:, g] \cdot V_B[:, g] \cdot V_C[:, g] \cdot P_D[:, g]$ ▷ $(n',)$

$W[:, c] \leftarrow W[:, c] \ \times\ |S_k^{(c)}|$

end for

end for

▷ Categorical sampling from noisy projection distribution

$\mathbf{p} \leftarrow W \,/\, \mathrm{rowsum}(W)$ ▷ $P(\mathbf{m}_c \mid \mathbf{f})$ for each shot

results$[\overline{\mathcal{I}_0}] \leftarrow$ `categorical`$(\log \mathbf{p},\ \mathrm{key})$

end for

return results

end function

Complexity. The baseline pipeline requires six matrix multiplications of size $(G{\cdot}T,\ P){\times}(P,\ nM)$ per sub-component per batch, which is the dominant cost. The split f/m decomposition (see section 9.4.5) replaces these with six multiplications of size $(G{\cdot}T,\ n_f){\times}(n_f,\ n)$, reducing the matmul cost by a factor of $M = 32$. The m-contributions are six multiplications of size $(G{\cdot}T,\ n_m){\times}(n_m,\ M)$, precomputed once at compile time. The per-combo element-wise operations cost $\mathcal{O}(n \cdot M \cdot G \cdot T_{\max})$, bounded in memory by $(n,\ G,\ T_{\max})$ tensors per combo iteration. The total per-batch cost is $\mathcal{O}(n \cdot G \cdot T \cdot n_f + n \cdot M \cdot G \cdot T)$ versus the baseline $\mathcal{O}(n \cdot M \cdot G \cdot T \cdot P)$.

B Accelerated Sampling Pipeline Pseudocode

Algorithm 4 extends the per-batch sampling of algorithm 3 with sparse geometric channel sampling (section 10.1), unique-pattern deduplication with a persistent cross-batch evaluation cache (section 10.2), and scan-based combo evaluation (section 10.3). The compile-time setup (Algorithm 2) is unchanged except for precomputing per-channel fire probabilities and conditional CDFs, the noiseless cache distributions, and D-term lookup tables.

Algorithm 4 Accelerated per-batch sampling with sparse geometric channel sampling, cross-batch deduplication, and scan-based evaluation.

Input: Compiled program, batch size B, PRNG key, per-channel ($p_{\text{fire},j}$, cond_cdf_j, xor_pattern_j), noiseless cache $\hat{\mu}$, persistent eval cache $\mathcal{C}$
Output: Sampled detector outcomes for B shots

function SampleBatchFast(program, B, key)

▷ **Phase 1: Sparse geometric channel sampling + noiseless cache**

$\mathbf{F} \leftarrow \mathbf{0}^{B \times n_f}$; $\text{noisy\_mask} \leftarrow \mathbf{0}^B$
for $j = 1$ **To** N_{ch} **do** ▷ geometric skip: only visit fire events
  $\text{skips} \leftarrow \text{Geometric}(p_{\text{fire},j})$ ▷ draw skip distances
  $\text{pos} \leftarrow \text{CumSum}(\text{skips}) - 1$; keep pos $< B$
  if $K_j = 2$ **then** ▷ binary channel: single non-identity outcome
    $\mathbf{F}[\text{pos}] \oplus= \text{xor\_pattern}_j$
  else ▷ multi-outcome: conditional sampling
    $\mathbf{u} \leftarrow \text{Uniform}(0,1)^{|\text{pos}|}$
    $\text{idx} \leftarrow \text{SearchSorted}(\text{cond\_cdf}_j, \mathbf{u})$
    $\mathbf{F}[\text{pos}] \oplus= \text{xor\_pattern}_j[\text{idx}]$
  end if
  $\text{noisy\_mask}[\text{pos}] \leftarrow 1$
end for

$\text{results} \leftarrow \text{SampleNoiseless}(\hat{\mu}, B)$ ▷ NumPy PCG64, column-by-column
$\mathcal{I}_{\text{noisy}} \leftarrow \{i : \text{noisy\_mask}[i] = 1\}$
$\mathbf{F}' \leftarrow \mathbf{F}[\mathcal{I}_{\text{noisy}}, :]$ ▷ (n', n_f), $n' = |\mathcal{I}_{\text{noisy}}|$

▷ **Phase 2: Deduplication + persistent cache + scan evaluation**
▷ *Step 1: Hash f-patterns to int64 scalars (NumPy, outside JIT)*
$\text{sel} \leftarrow$ f-parameter indices for enum component ▷ $|\text{sel}| = n_f = 37$
$\mathbf{h} \leftarrow \mathbf{F}'[:, \text{sel}] \cdot (2^0, 2^1, \ldots, 2^{n_f-1})$ ▷ $(n',)$ int64 via BLAS matmul

▷ *Step 2: Find unique patterns + cross-batch cache lookup*
$(\mathbf{h}_{\text{uniq}}, \sigma) \leftarrow \text{NumpyUnique}(\mathbf{h})$ ▷ U unique hashes + inverse map
$\text{new} \leftarrow \{i : \mathbf{h}_{\text{uniq}}[i] \notin \mathcal{C}\}$ ▷ only genuinely new patterns

▷ *Step 3: Decode new patterns, pad, and evaluate*
$\tilde{\mathbf{F}} \leftarrow \left((\mathbf{h}_{\text{uniq}}[\text{new}] \gg [0, \ldots, n_f - 1]) \;\&\; 1\right)$
$\hat{U} \leftarrow 2^{\lceil \log_2 |\text{new}| \rceil}$; pad $\tilde{\mathbf{F}}$ to $(\hat{U}, n_f)$

▷ *Step 4: Split f/m evaluation via `jax.lax.scan` (JAX JIT)*
$\mathbf{J} \leftarrow \mathbf{1}^{\hat{U} \times M}$ ▷ joint magnitudes for new patterns
for $k = 1$ **To** K **do** ▷ each sub-component
  for $\tau \in \{D_\alpha, D_\beta, A, B, C_\psi, C_\phi\}$ **do**
    $R_f^{(\tau)} \leftarrow \left(\text{reshape}(B_f^{(\tau)}) \times \tilde{\mathbf{F}}_k{}^\top\right) \bmod 2$ ▷ f-only row sums, once per k
  end for
  D-term lookup: precompute $d_{00}, d_{01}, d_{10}, d_{11}$ per (G_k, T_D) position
  ▷ combo loop via `jax.lax.scan` (single compiled body)
  for $c = 1$ **To** M **do**
    $R^{(\tau)} \leftarrow (R_f^{(\tau)} + R_m^{(\tau,k)}[c]) \bmod 2$ for each τ
    D-terms: select from $\{d_{00}, d_{01}, d_{10}, d_{11}\}$ by parity bits
    $S_k^{(c)} \leftarrow \text{EvalTerms}(\ldots)$; $\mathbf{J}[:, c] \leftarrow \mathbf{J}[:, c] \times |S_k^{(c)}|$
  end for
end for
Store $\mathbf{J}$ in persistent cache $\mathcal{C}$ for new hashes

▷ *Step 5: Assemble all magnitudes from cache + categorical sampling*
$\mathbf{J}_{\text{all}} \leftarrow$ look up $\mathcal{C}[\mathbf{h}_{\text{uniq}}]$ for all U patterns
$\mathbf{P}_i \leftarrow \mathbf{J}_{\text{all}}[\sigma(i), :]$ for each noisy shot i
$\mathbf{p}_i \leftarrow \mathbf{P}_i$ / rowsum($\mathbf{P}_i$)
$\mathbf{c} \leftarrow \texttt{categorical}(\log \mathbf{p}, \text{key})$
$\text{results}[\mathcal{I}_{\text{noisy}}] \leftarrow \mathbf{m_c}$

return results
end function

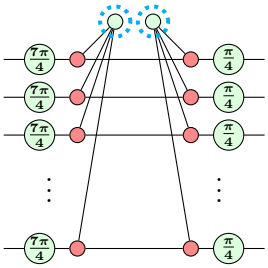
Complexity. Let η denote the noiseless fraction and U the number of unique fault patterns among the $(1-\eta)B$ noisy shots per batch. Phase 1 costs $O(N_{\text{ch}} \cdot B \cdot \bar{p}_{\text{fire}})$ per batch via geometric skip: only fire events incur random draws, compared with $O(N_{\text{ch}} \cdot B)$ for inverse CDF or $O(N_{\text{ch}} \cdot \bar{K} \cdot B)$ for Gumbel-Max. Phase 2 costs $O(B \log B)$ for hashing and deduplication, plus $O(\hat{U}_{\text{new}} \cdot K \cdot M \cdot G \cdot T_{\max})$ for the split f/m evaluation on genuinely new patterns only, where $\hat{U}_{\text{new}}$ is the number of patterns not

found in the persistent cache. The total number of evaluations across all batches is bounded by the number of distinct patterns in the entire run (∼1,000 at $p = 5 \times 10^{-4}$), rather than $U \times n_{\text{batches}}$ as in within-batch dedup alone.

Random number budget. The sparse geometric sampler generates $\sim 2N_{\text{ch}}B\bar{p}_{\text{fire}}$ random numbers per batch (one geometric draw per skip plus one conditional uniform per fire event). At $p = 5 \times 10^{-4}$ with $\bar{p}_{\text{fire}} \approx 0.001$, this amounts to ∼500K draws versus $N_{\text{ch}} \cdot B \approx 2.6 \times 10^8$ for inverse CDF or $N_{\text{ch}} \cdot \bar{K} \cdot B \approx 1.0 \times 10^9$ for Gumbel-Max. The noiseless cache adds $n_{\text{binary}} + n_{\text{categorical}}$ draws per batch (28 Bernoulli +1 categorical for this circuit, generated via NumPy PCG64). Phase 2 adds one categorical draw per noisy shot.

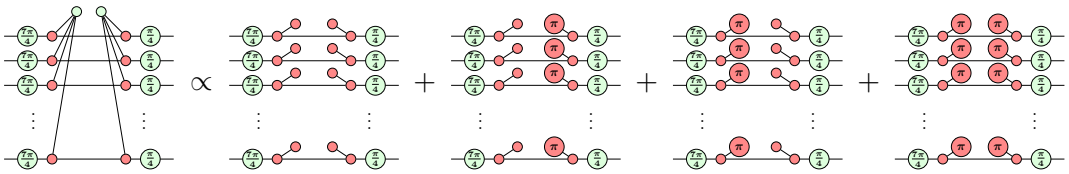
C Cutting the basic flag-free double checking circuit

The double checking sub-circuit dominates the T-count in cultivation circuits. Essentially, they are a series of n controlled-(H_{XY}) gates applied to qubits on a n-GHZ state as control and qubits on the injected colour code as target. Taking the double checking circuit from [1], removing all the flag like checks and allow $n \geq 2$ to be an arbitrary integer, we arrive at the following circuit (with n total inputs/outputs):



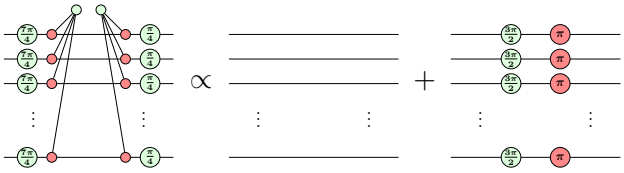
(26)

If we were to make two cuts at the two Z-spiders inside the cyan dotted circles via the cutting decompositions [2, 5, 10], we will have the following $2^2 = 4$ terms stabiliser decomposition:



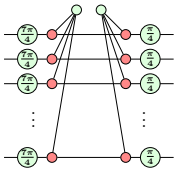
(27)

With a bit of simplifications and collecting terms, we arrive at:



(28)

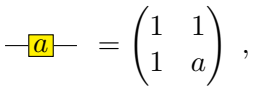
this implies



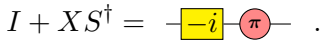
$$\propto I^{\otimes n} + (XS^\dagger)^{\otimes n} \,. \tag{29}$$

The basic flag-free double checking circuit can be written as a sum of 2 Clifford ZX-diagrams. Similar arguments can be made about the double checking circuit with Pauli errors and different measurement values. This suggest that the difficulty in finding stabiliser decomposition for such circuits from [1] maybe due to the presence of additional flag-like structures, leading to higher number of terms (4 and 8 terms) as we see in the main text and from [2].

Another potentially useless observation, $I + XS^\dagger = \begin{pmatrix} 1 & -i \\ 1 & 1 \end{pmatrix}$, which can be related to the Hadamard box:



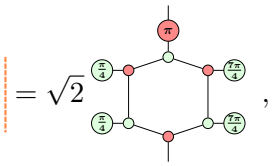
[32] from ZH-calculus via



We wonder if $(I + XS^\dagger)^{\otimes n} = I^{\otimes n} + (XS^\dagger)^{\otimes n} - \sum_j (\text{cross terms})_j$ admits a nice interpretation in ZH-calculus. Probably not useful.

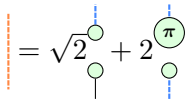
D An observation

We noticed that the orange dotted (star) edge [17] defined as:

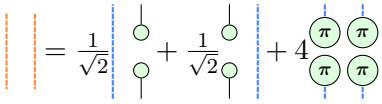


(30)

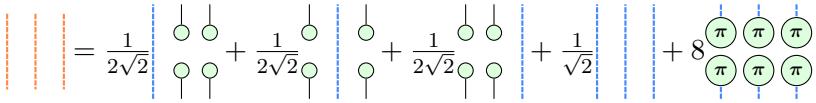
bears resemblance to the double checking circuits. This ZX-diagram has some very nice stabiliser decompositions[10].



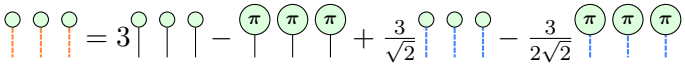
(31)



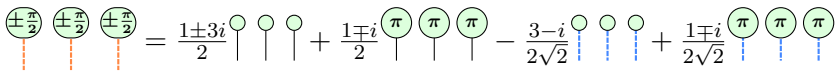
(32)



(33)



(34)



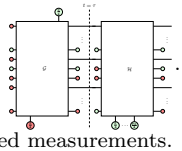
(35)

We wonder if these can be adapted to reduce the number of terms needed to simulate magic state cultivation even further via re-write etc. To any potential ZX-calculus experts, please do let us know.

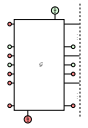
[10] Tikz figures taken from [17].

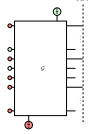
E State sampling

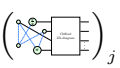
Algorithm 5 Sample measurements.

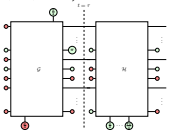
Input: Original errored ZX-diagram $g =$  .

Output: g_{meas}, ZX-diagram with sampled measurements.

function SubDiagram(g, τ)
Obtain sub-diagram up to the first measurement time slice ($t = \tau$): h.
return $h =$ 
end function

function StripMeasSpiders(h, τ)
Strip measurement spiders in ZX-diagram h at timeslice $t = \tau$.
return $h =$ 
end function

function StabiliserDecompose(h)
Perform stabiliser decomposition (cutting and further secondary decompositions) on ZX-diagram h.
return $s = \sum_{j=1}^{\chi} a_j \left($  $\right)_j$
end function

function SampleAndReplaceMeasSpiders(s, g)
Sample decomposed sum of Clifford ZX-diagrams, only at the stripped measurement spider legs. Obtaining measurement sample: $\vec{M} = (\pi, 0, 0, \pi, \dots)$ for example. Replace original ZX-diagram g with measurement results.
return $g_{\text{meas}} =$ 
end function

function Main(g)
for $\tau \in \{1, 2, 3, ..., \text{End}\}$ **do**
$h \leftarrow$ SubDiagram(g, τ)
$h \leftarrow$ StripMeasSpiders(h, τ)
$s \leftarrow$ StabiliserDecompose(h)
$g \leftarrow$ SampleAndReplaceMeasSpiders(s, g)
end for
return g (with measurement spider legs sampled)
end function

$g_{\text{meas}} \leftarrow$ Main(g)

F A few detecting regions for the $d = 3$ circuit with degenerate injection

Following the same detector numbering from [19], here are a few detectors for the $d = 3$ circuits with degenerate injection.

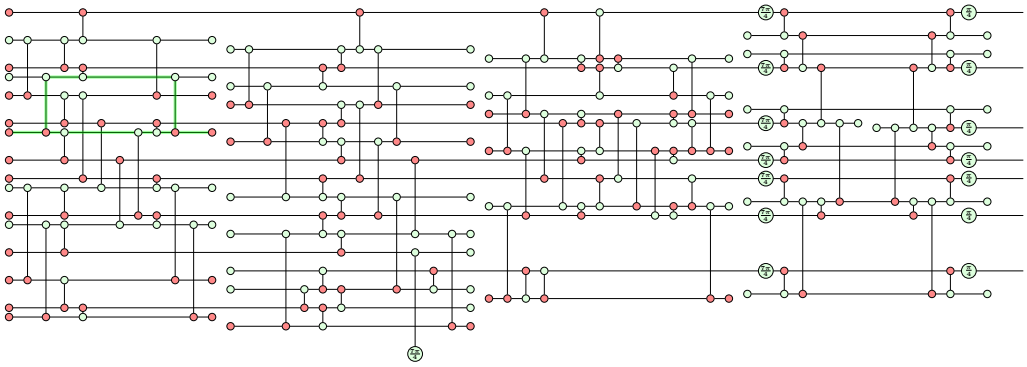


Figure 13: Detecting region/closed Pauli web/detector 0.

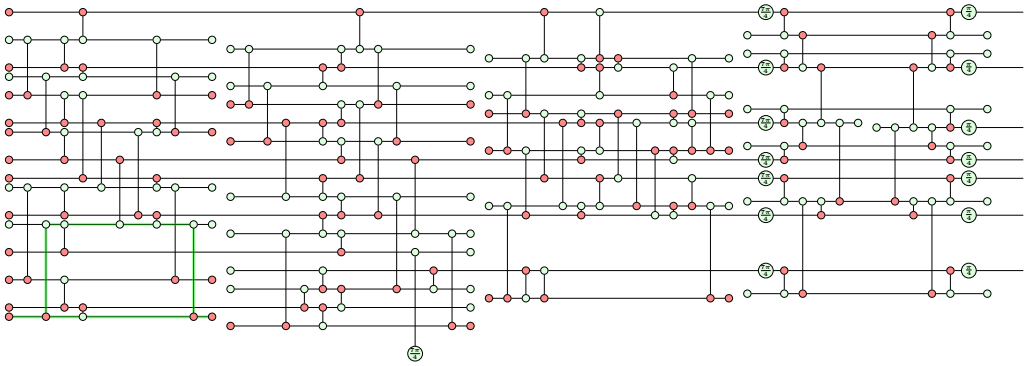


Figure 14: Detecting region/closed Pauli web/detector 2.

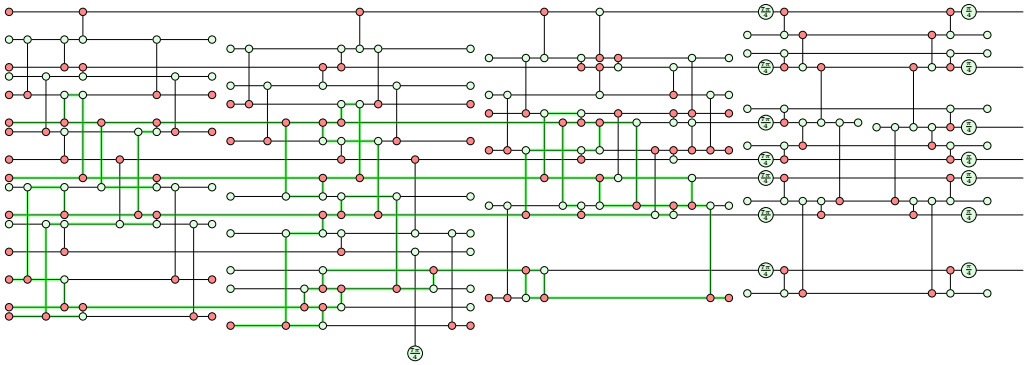


Figure 15: Detecting region/closed Pauli web/detector 18.

G A few detecting regions for the $d = 5$ circuit with degenerate injection

Again, following the same detector numbering from [19], here are 6 detectors for the $d = 5$ circuit with degenerate injection. The rest are available in the supplementary tikz folder `d5_det_webs`.

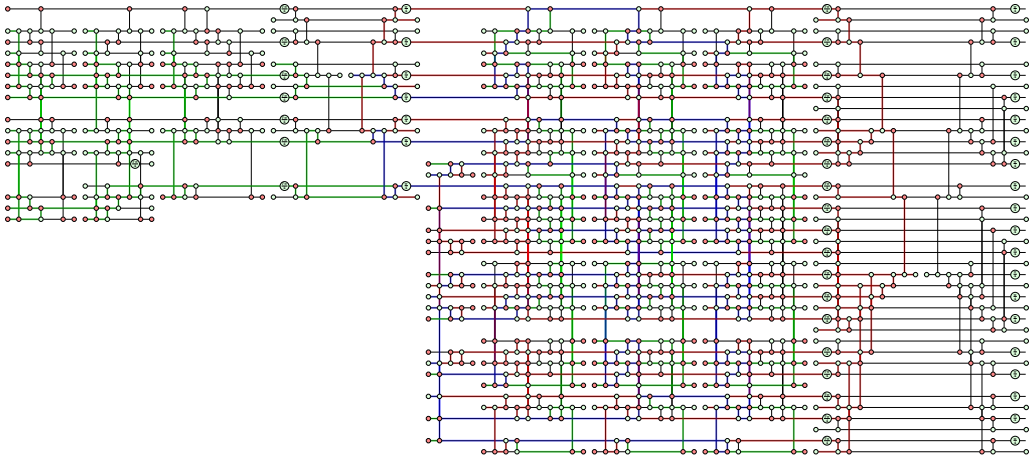


Figure 16: Detector 81.

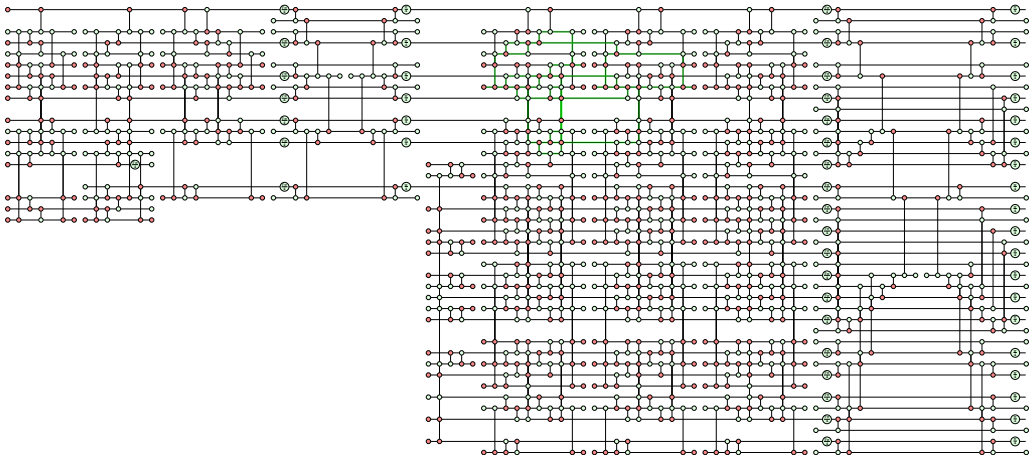


Figure 17: Detector 48.

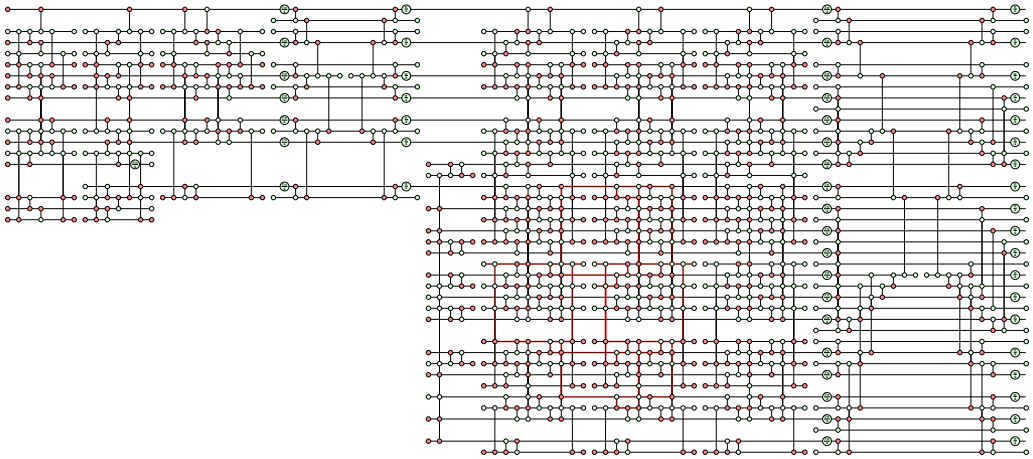


Figure 18: Detector 55.

H Sketch example

The figure illustrates the sample measurements approach applied to the initial stages of the $d = 3$ cultivation circuit, showing the contraction up to the first measurement layer and subsequent stabiliser decomposition.