



Title	Efficient Preparation and Consumption of Magic States for Fault-Tolerant Quantum Computing
Author(s)	平野, 裕
Citation	大阪大学, 2026, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/105547
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

Efficient Preparation and Consumption of Magic States for Fault-Tolerant Quantum Computing

YUTAKA HIRANO

MARCH 2026

Efficient Preparation and Consumption of Magic States for Fault-Tolerant Quantum Computing

A dissertation submitted to
THE GRADUATE SCHOOL OF ENGINEERING SCIENCE
The University of Osaka

in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY IN SCIENCE

BY
YUTAKA HIRANO

MARCH 2026

Abstract

Quantum computation promises exponential or polynomial speedups over classical computation for a wide range of problems, including integer factoring, quantum chemistry, and linear-system solving. However, current quantum devices remain too noisy to execute complex algorithms reliably. Overcoming this noise and achieving quantum advantage require a fault-tolerant quantum computer (FTQC). The resource overhead associated with fault-tolerant operation is substantial, and reducing this overhead remains one of the central challenges.

A major source of this overhead arises from the implementation of non-Clifford gates, which are essential for universal quantum computation. A non-Clifford gate such as the T gate is implemented using a special quantum state known as a magic state, which is prepared via a process called magic state distillation. Magic state distillation is resource-intensive and remains a major bottleneck in FTQC. Hence, reducing the distillation overhead is necessary to lower the overall FTQC overhead.

The large overhead of magic state distillation has significantly influenced FTQC architectures. In particular, it has motivated compilation schemes optimized under the constraint of expensive magic state distillation. While reducing the distillation overhead is crucial, it is not sufficient; lowering the overall FTQC overhead also requires revisiting design choices guided by this constraint. Therefore, efficient methods are required not only for preparing magic states but also for consuming them.

In this thesis, we reduce the overhead of fault-tolerant quantum computation by focusing on the preparation and consumption of magic states. In contrast to traditional logical-level distillation, physical-level distillation allows direct implementation with physical gates and has the potential to drastically reduce the distillation overhead. However, existing physical-level distillation protocols, such as zero-level distillation, cannot directly yield high-fidelity magic states. We propose two magic state distillation protocols that use physical-level distillation techniques to efficiently prepare high-fidelity magic states. The first protocol is a two-level distillation protocol using physical-level distillation as the first-level stage. The second protocol utilizes lattice surgery to reduce the spacetime overhead of a physical-level distillation protocol. The former targets a high-fidelity regime, whereas the latter targets an intermediate-fidelity regime—lower than the former, but higher than zero-level distillation.

Next, we investigate FTQC architectures that can efficiently consume the magic states prepared by these protocols. Because magic state distillation is probabilistic, occasional failures can delay computation. Moreover, physical-level distillation protocols generally exhibit higher failure probabilities than logical-level ones. This delay is amplified by parallel execution, making it an

important consideration for FTQC using physical-level distillation protocols. We therefore propose a scheme to mitigate the negative performance impact caused by magic state distillation failures by introducing a pool of magic states for each distillation factory.

The proposed distillation protocols allow the simultaneous preparation of many magic states. Leveraging this capability, we propose an FTQC compilation scheme optimized for highly parallel quantum algorithms. By performing magic state distillation locally and consuming magic states in parallel, the scheme enables highly parallel execution and consequently reduces the computational time overhead.

Overall, this thesis presents a unified approach to reducing the overhead of fault-tolerant quantum computation. By improving both the preparation and consumption of magic states, it advances the design of resource-efficient FTQC architectures. These results demonstrate that efficient magic state preparation motivates efficient magic state consumption, and vice versa. Together, they bring us closer to the practical realization of fault-tolerant quantum computers.

Contents

1	Introduction	1
1.1	Background	1
1.2	Overview	3
1.3	Outline	6
2	Preliminaries	8
2.1	Quantum computation	8
2.1.1	Quantum state	8
2.1.2	Quantum gate	8
2.1.3	Quantum circuit	10
2.1.4	Quantum computer hardware	10
2.2	Quantum error correction	11
2.2.1	Stabilizer formalism	11
2.2.2	Decoding	13
2.2.3	Fault-tolerant quantum computation	14
2.2.4	Surface code	15
2.2.5	Lattice surgery	18
2.2.6	Complementary gap	19
2.3	Compilation schemes for FTQC	20
2.3.1	FTQC compilation	20
2.3.2	Sequential Pauli-based computation	23
2.4	Magic state distillation	25
2.4.1	15-to-1 distillation	27
2.4.2	(15-to-1)×(15-to-1) distillation	31
2.4.3	Zero-level distillation	32
2.4.4	Magic state cultivation	32
2.5	Example quantum circuits for benchmarking	34
2.5.1	Random circuit sampling	34
2.5.2	SELECT oracle	34
2.5.3	Hamiltonian simulation with Trotterization	36

3	Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States	37
3.1	Introduction	37
3.2	(0+1)-level distillation	39
3.2.1	Zero-level distillation integration	39
3.2.2	Teleportation	40
3.2.3	Shortening the height of the ancilla region	41
3.2.4	(0+1)-level distillation	42
3.3	Performance evaluation	44
3.3.1	Spacetime overhead	44
3.3.2	Case study for massively parallel magic state consumption	46
3.4	Conclusion	50
4	Efficient Magic State Cultivation with Lattice Surgery	51
4.1	Introduction	51
4.2	Magic state cultivation with lattice surgery	53
4.2.1	Lattice surgery	54
4.2.2	Code expansion	56
4.2.3	Lookup table	57
4.3	Numerical simulation	59
4.3.1	Simulation settings	59
4.3.2	Results	59
4.4	Conclusion	63
5	MagicPool: Dealing with Magic State Distillation Failures on Large-Scale Fault-Tolerant Quantum Computer	65
5.1	Introduction	65
5.2	Parallel execution	67
5.3	Impact of RUS failures	68
5.4	Dealing with distillation failures	70
5.5	Performance evaluation	73
5.5.1	Random quantum circuit simulation	73
5.5.2	Dist-SELECT simulation: scheduling	76
5.5.3	Dist-SELECT simulation: execution	78
5.5.4	Dist-SELECT simulation: parameter settings	79
5.5.5	Dist-SELECT simulation: results	80
5.6	Conclusion	81
6	Locality-Aware Pauli-Based Computation for Local Magic State Preparation	83
6.1	Introduction	83
6.2	Locality-aware Pauli-based computation	85
6.2.1	Overview of the scheme	86
6.2.2	Related studies	87

6.2.3	Magic state distillation protocol	88
6.2.4	Quantum ISA and its implementation	88
6.2.5	Transpilation to the quantum ISA	90
6.3	Performance evaluation	91
6.3.1	Scheduler implementation	91
6.3.2	Simulator implementation	92
6.3.3	Parameter settings	93
6.3.4	Results	93
6.4	Conclusion	95
7	Conclusion	97

Chapter 1

Introduction

1.1 Background

In 1981, Feynman [1] pointed out the difficulty of simulating quantum systems efficiently and envisioned a new type of computer that obeys the laws of quantum mechanics, which he called a *quantum computer*. Later, Deutsch proposed a universal quantum computer [2], and since then, many applications have been proposed, such as integer factoring [3], simulating quantum systems [4], database search [5], quantum chemistry [6], and large-scale linear-system solving [7].

Various hardware platforms have been proposed for realizing quantum computers. The trapped-ion platform offers long coherence times and high gate fidelities but suffers from slow gate speeds and scalability challenges. Superconducting devices, in contrast, offer fast gate speeds. This platform imposes a connectivity restriction: only neighboring qubits can interact. The neutral-atom platform offers long coherence times and movable qubits, which allow more flexible connectivity, but it operates with relatively slow gate speeds.

Although hardware progress has been rapid, including the demonstration of quantum computational supremacy on superconducting devices [8], none of these existing quantum computers is capable of executing a quantum algorithm with a theoretically proven quantum speedup such as integer factoring. This limitation arises because currently available quantum computers are too noisy to run complex quantum algorithms. We therefore need a *fault-tolerant quantum computer* (FTQC) [9] based on *quantum error correction* (QEC) [10]. Quantum error correction has been experimentally demonstrated on superconducting [11] and neutral-atom [12] devices. However, building a large-scale FTQC involves considerable resource overhead, making the reduction of this overhead a central challenge.

An FTQC is a virtual quantum computer that, running on a noisy physical device, simulates the execution of a noise-free quantum circuit. To this end, qubits in the original circuit are encoded in a QEC code, and each noiseless gate (*logical gate*) in the original circuit is translated into a *fault-tolerant gadget* built from noisy gates (*physical gates*) in a fault-tolerant manner. We call this translation process *FTQC compilation*¹. While fault-tolerant implementations of Clifford gates are

¹In this thesis, we include arbitrary-angle Z rotations in the logical gate set. Accordingly, gate decomposition, which translates an arbitrary-angle Z rotation into a sequence of H , S , and T gates, is part of the compilation process (see

relatively inexpensive, Clifford gates alone are not sufficient for universal quantum computation [13]. For universal quantum computation, at least one non-Clifford gate, such as the T gate, is required. However, non-Clifford gates are much more expensive to implement fault-tolerantly than Clifford gates. A widely used approach for the fault-tolerant implementation of non-Clifford gates is to use special quantum states called *magic states*, which are obtained via a process known as *magic state distillation* [14]. Although other methods for the fault-tolerant implementation of non-Clifford gates exist, such as code switching [15–17], magic state distillation is regarded as one of the most promising approaches [18] in terms of computational cost. Nevertheless, magic state distillation is resource-intensive compared to Clifford gates. For example, Jones *et al.* [19] reported that approximately 90% of physical qubits were devoted to distillation in their resource estimation of factoring a 1024-bit integer (4.2×10^8 physical qubits were devoted to distillation, whereas 3.8×10^7 physical qubits were used as application qubits). Consequently, many distillation protocols have been proposed [14, 20–29], but distillation remains a major bottleneck in fault-tolerant quantum computation.

To understand the origin of the large overhead associated with magic state distillation, we review the historical development of magic state distillation protocols. Bravyi *et al.* [14] proposed the 15-to-1 distillation protocol, which exploits T 's transversality on the $[[15, 1, 3]]$ code. Earlier, predating the term “magic state distillation”, Knill *et al.* [20] proposed a protocol based on the Hadamard test of the H gate, relying on H 's transversality on the $[[7, 1, 3]]$ code. These studies formulated magic state distillation as a quantum algorithm built from ideal Clifford gates with noisy inputs, and this formulation was followed by subsequent studies [21, 22, 24–26]. Running such algorithms on an FTQC requires protecting each gate against noise using QEC, incurring a significant cost.

Goto [23] proposed a technique that we call *physical-level distillation*. In contrast to the above formulation, which we call logical-level distillation, the distillation circuit operates with noisy Clifford gates. Specifically, the circuit detects any single gate error, thereby achieving a logical error probability of $\mathcal{O}(p_{\text{phys}}^2)$, where p_{phys} is the physical error probability. To this end, the distillation protocol consists of a Knill-style Hadamard test with a flag qubit. Its physical-gate implementation significantly reduces the distillation overhead.

Goto's protocol requires all-to-all connectivity. Chamberland *et al.* [28] and Itogawa *et al.* [29] proposed physical-level distillation protocols that work with two-dimensional (2D) nearest-neighbor connectivity. The latter, known as zero-level distillation, operates on the simpler square-grid connectivity and yields a magic state encoded in a surface code [30, 31], achieving a logical error probability of 10^{-4} for a realistic $p_{\text{phys}} = 10^{-3}$ setting. Thus, although it cannot directly yield high-fidelity magic states, physical-level distillation has the potential to drastically reduce the distillation overhead.

Retrospectively, research on magic state distillation predominantly focused on logical-level distillation, as it provides a theoretically clean abstraction and facilitates the discussion of asymptotic performance. For example, recursive constructions were extensively analyzed in terms of the number of noisy magic states required to produce a magic state with a given target fidelity [21, 22, 32]. In

contrast, physical-level distillation applies only at the physical level and offers only constant-factor improvements in the asymptotic regime; it was therefore regarded as less impactful from an asymptotic complexity perspective.

In recent years, however, it has become apparent that the performance improvements offered by physical-level distillation are substantial, especially for near-term applications. As hardware development progresses and physical error probabilities decrease, the required number of distillation recursion levels is reduced, amplifying the impact of constant-factor improvements. Moreover, advances in architectural proposals and resource estimation methodologies [26, 33] enable more precise performance evaluation under realistic hardware constraints, beyond the number of noisy magic states required.

The large overhead of magic state distillation has significantly influenced FTQC architectures. In particular, it has motivated compilation schemes optimized under the constraint of expensive magic state distillation. Sequential Pauli-based computation (SPC) [26] is one such compilation scheme. It transpiles a sequence of Clifford+ T gates into multi-qubit Pauli measurements, each of which consumes a magic state. The resulting sequence of multi-qubit Pauli measurements is difficult to execute in parallel, but this is not problematic when the magic state distillation overhead is large. If the magic state distillation overhead were to decrease, such design choices should be revisited. This example illustrates that reducing the total overhead of FTQC requires efficient methods not only for preparing magic states but also for consuming them. Akahoshi *et al.* [34–36] proposed the SpaceTime efficient Analog Rotation (STAR) architecture, which directly prepares quantum states needed for rotation gates. Although this direct preparation has limitations—for example, it suffers from large logical error probabilities for large-angle rotations—this architecture offers performance benefits for applications that extensively use small-angle rotation gates over conventional Clifford+ T -based architectures, under the assumption of expensive T magic state distillation. This is another example of an FTQC architecture optimized under the constraint of expensive magic state distillation.

These observations motivate the study of magic state distillation protocols and architectures that minimize the overall FTQC overhead, which is the central focus of this thesis.

1.2 Overview

In this thesis, we reduce the overhead of fault-tolerant quantum computation by focusing on the preparation and consumption of magic states. We target the superconducting platform, which requires 2D nearest-neighbor connectivity, because it is one of the most advanced quantum computing hardware platforms. Quantum error correction using surface codes has been experimentally demonstrated on this platform [11], and from a theoretical perspective, the surface-code-based architecture has matured sufficiently to enable architecture-level design discussions for large-scale fault-tolerant quantum computation [26, 33, 34, 37]. First, we propose two efficient magic state distillation protocols that use physical-level distillation.

Zero-level distillation [29] offers significantly smaller distillation overhead compared to traditional logical-level distillation protocols, but it cannot directly generate high-fidelity magic

states since it only reduces the logical error probability of the magic state quadratically. We propose *(0+1)-level distillation*, a two-level magic state distillation protocol that uses zero-level distillation as the first-level distillation protocol to yield high-fidelity magic states. Through numerical simulations, we find that the proposed protocol yields high-fidelity magic states and reduces the distillation overhead compared to existing two-level distillation protocols. We also demonstrate that this efficiency gain in end-to-end performance is substantial, namely, using the proposed protocol reduces the total spatial cost by more than 30%.

Magic state cultivation [38] is a refinement of zero-level distillation and is another attempt to generate high-fidelity magic states by applying Hadamard tests repeatedly and extensively utilizing postselection. It prepares a magic state encoded in a 2D color code [39], then transforms it into the grafted code, another QEC code compatible with matching QEC decoders. However, due to its high-weight stabilizer checks, the grafted code is less space-efficient than the rotated surface code [40]. We propose *magic state cultivation with lattice surgery*, a physical-level distillation protocol that utilizes lattice surgery [41, 42], as in zero-level distillation, to transfer the magic state from the color code to the rotated surface code to reduce the distillation overhead. The proposed protocol achieves a logical error probability for the resulting magic state comparable to that of magic state cultivation, while requiring about half the spacetime overhead.

Then, we discuss FTQC architectures that can consume magic states efficiently. Existing FTQC architectures are optimized under the constraint of expensive magic state distillation. The above distillation protocols drastically reduce the distillation overhead, but revisiting design choices related to this constraint is necessary to translate that reduction into a reduction in the overall FTQC overhead.

Magic state distillation is a probabilistic process and can fail. When it fails, the system retries distillation and the consumer of the magic state needs to wait. This delay, called a runtime delay, is amplified by parallel processing. Moreover, physical-level distillation protocols generally exhibit higher failure probabilities than logical-level ones. Thus, this problem becomes more significant when we prepare and consume many magic states simultaneously using physical-level distillation protocols. We propose a scheme to mitigate the negative performance impact caused by magic state distillation failures: specifically, we maintain a pool of magic states. We quantify the performance impact of distillation failures and evaluate the effectiveness and cost of pooling through numerical simulations. The results show that distillation failures can cause substantial runtime delays, which pooling effectively reduces while adding only a small spatial overhead.

SPC [26] uses a layout that separates the computation area from the distillation area. In this layout, executing a T gate involves three steps: (i) distilling a magic state in the distillation area, (ii) transferring it to a qubit adjacent to the target data qubit in the computation area, and (iii) performing gate teleportation. This layout has been widely adopted in resource estimation and compilation studies for FTQC with nearest-neighbor connectivity [33, 43–47], but it is an obstacle to the simultaneous consumption of many magic states due to the magic state transfer cost. We propose *locality-aware Pauli-based computation*, an FTQC compilation scheme optimized for highly parallel quantum algorithms. In contrast to SPC, which sequentially executes Pauli measurements, the proposed scheme supports parallel Pauli measurements. To enable simultaneous

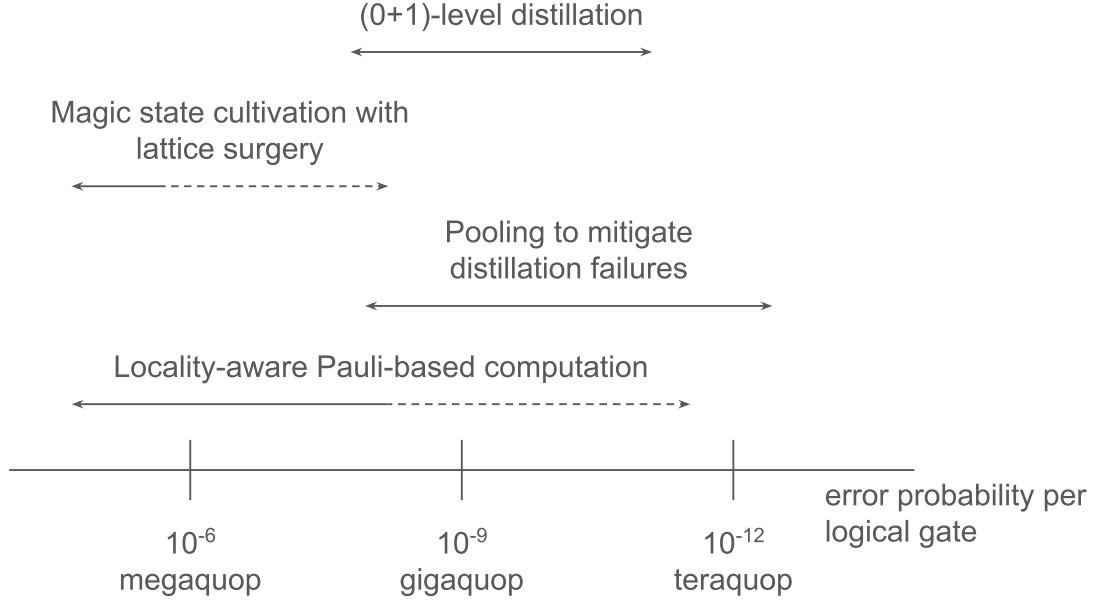


Figure 1.1: Target computation size of each contribution in this thesis. Solid ranges represent the target computation sizes, which may extend to dashed ranges once the future directions discussed in this thesis are resolved.

consumption of many magic states required by parallel Pauli measurements, the scheme employs local magic state distillation, under the assumption of efficient magic state distillation. By performing magic state distillation locally and consuming magic states in parallel, the scheme enables highly parallel execution and consequently reduces the computational time overhead. Through numerical simulations, we demonstrate that the enhanced locality enables greater parallelism than existing parallel-computing schemes based on the conventional layout. We also show that the proposed scheme exhibits favorable scaling in execution time as the system size increases.

Figure 1.1 illustrates the target computation sizes of the contributions in this thesis as a function of the error probability per logical gate. There, solid ranges represent the target regimes addressed in this thesis, while dashed extensions represent regimes that may become reachable once the future directions discussed in this thesis are resolved. We describe these regimes using Preskill’s notions of megaquop, gigaquop, and teraquop [48], which correspond to an error probability per logical gate of order 10^{-6} , 10^{-9} , and 10^{-12} , respectively. As shown in the figure, (0+1)-level distillation mainly targets the gigaquop regime, magic state cultivation with lattice surgery targets the megaquop regime, pooling to mitigate distillation failures spans the gigaquop–teraquop regime, and locality-aware Pauli-based computation covers the megaquop–gigaquop regime. Overall, these targets correspond to fault-tolerant quantum computers that can be realized in the relatively near term.

Taken together, this thesis proposes efficient methods for magic state preparation and consumption. These two components are closely related rather than independent. Efficient magic state preparation motivates efficient magic state consumption, and vice versa. Together, they reduce the overall computation overhead and contribute to the realization of FTQC.

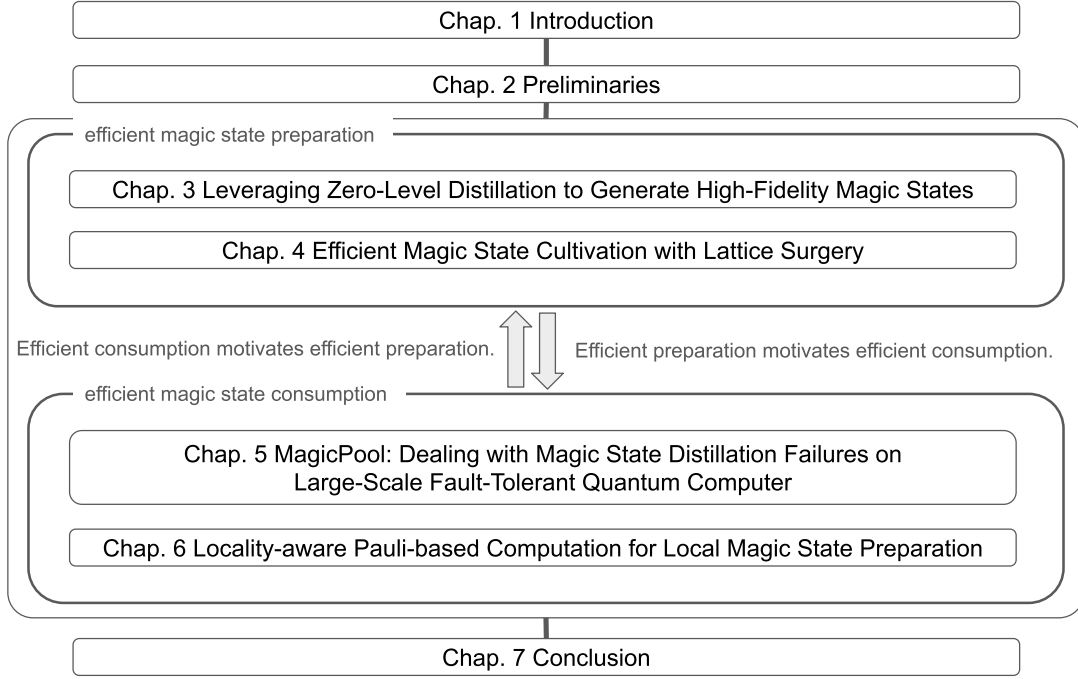


Figure 1.2: Structure of this thesis.

1.3 Outline

The rest of this thesis is structured as follows, as illustrated in [Figure 1.2](#). As shown in the figure, efficient magic state preparation motivates efficient magic state consumption, and vice versa. Together, they reduce the overall computational overhead and contribute to the realization of FTQC.

Chapter 2: Preliminaries

In this chapter, we introduce the basics of quantum computation and fault-tolerant quantum computation relevant to this thesis. We also introduce FTQC compilation schemes and magic state distillation—the focus of this thesis—as well as example algorithms used for performance evaluation.

Chapter 3: Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States

We propose *(0+1)-level distillation*, a two-level distillation protocol that uses zero-level distillation as the first-level distillation. We find that the proposed protocol yields high-fidelity magic states while reducing the distillation overhead compared to existing two-level distillation protocols. We also demonstrate that this efficiency gain in end-to-end performance is substantial. This chapter is based on “**Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States,**” Hirano, Itogawa, Fujii, *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 843–853 [49].

Chapter 4: Efficient Magic State Cultivation with Lattice Surgery

We propose *magic state cultivation with lattice surgery*, a physical-level distillation protocol that uses lattice surgery and the rotated surface code to reduce the distillation overhead. It achieves a logical error probability for the resulting magic state comparable to that of magic state cultivation, while requiring about half the spacetime overhead. This chapter is based on “**Efficient magic state cultivation with lattice surgery,**” Hirano, Toshio, Itogawa, Fujii, *arXiv: 2510.24615* [50].

Chapter 5: MagicPool: Dealing with Magic State Distillation Failures on Large-Scale Fault-Tolerant Quantum Computer

We propose a scheme to mitigate the negative performance impact caused by magic state distillation failures: specifically, we maintain a pool of magic states. We quantify the performance impact of magic state distillation failures on applications and evaluate the effectiveness and cost of pooling through numerical simulations. The results show that distillation failures can cause substantial runtime delays, which pooling effectively reduces while adding only a small spatial overhead. This chapter is based on “**MagicPool: Dealing with Magic State Distillation Failures on Large-Scale Fault-Tolerant Quantum Computer,**” Hirano, Suzuki, Fujii, *arXiv: 2407.07394* [51].

Chapter 6: Locality-Aware Pauli-Based Computation for Local Magic State Preparation

We propose *locality-aware Pauli-based computation*, an FTQC compilation scheme optimized for highly parallel quantum algorithms. By performing magic state distillation locally and consuming magic states in parallel, the scheme enables highly parallel execution and consequently reduces the computational time overhead. Through numerical simulations, we demonstrate that the enhanced locality enables greater parallelism than existing parallel-computing schemes based on the conventional layout. We also show that the proposed scheme exhibits favorable scaling in execution time as the system size increases. This chapter is based on “**Locality-aware Pauli-based computation for local magic state preparation,**” Hirano, Fujii, *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 670–680 [52].

Chapter 7: Conclusion

We summarize the main contributions of this thesis and discuss potential directions for future work.

Chapter 2

Preliminaries

2.1 Quantum computation

2.1.1 Quantum state

For any positive integer n , we define the *state space* of an n -qubit system as a 2^n -dimensional complex Hilbert space $\mathbb{C}^{2^n}$. An n -qubit *quantum state* is a unit vector in the n -qubit state space. For example, a single-qubit quantum state $|\psi\rangle$ is expressed as

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle,$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Here, $|0\rangle$ and $|1\rangle$ are defined as

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

and the set $\{|0\rangle, |1\rangle\}$ forms an orthonormal basis of the single-qubit state space, called the Z basis. Another orthonormal basis, called the X basis, is given by $\{|+\rangle, |-\rangle\}$, where

$$|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad |-\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

More generally,

$$\left\{ |b_0 \cdots b_{n-1}\rangle = \bigotimes_{i=0}^{n-1} |b_i\rangle \mid \forall b_i \in \{0, 1\} \right\}$$

forms an orthonormal basis of the n -qubit state space. Analogous to the bit in classical computation, the qubit is the most basic unit of information in quantum computation.

2.1.2 Quantum gate

Quantum computers perform computation using *quantum gates* and *measurement operations*. We begin by defining quantum gates.

A quantum gate is given by a unitary operator acting on a finite number of qubits. Although there are infinitely many unitary operators, a quantum computer has a fixed set of quantum gates, called a *gate set*. A gate set is said to be *universal* if any unitary operator can be approximated to arbitrary accuracy using only the gates in the set.

We introduce basic quantum gates. Single-qubit Pauli operators (and hence Pauli gates) I , X , Y , and Z are defined as

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

More generally, an n -qubit Pauli operator is a tensor product of single-qubit Pauli operators acting on the n -qubit state space. The set of n -qubit Pauli operators, together with factors ± 1 and $\pm i$, forms a group and is denoted by $\mathcal{P}_n$. Clifford operators are defined as unitary operators that map a Pauli operator to a Pauli operator through conjugation:

$$\mathcal{C}_n := \{U \mid U\mathcal{P}_nU^\dagger = \mathcal{P}_n\}.$$

For example, the Hadamard gate H and the phase gate S are important single-qubit Clifford gates:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

The CX gate is a Clifford gate that acts on two qubits,

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The first qubit is called the control qubit, and the second qubit is called the target qubit. For universal quantum computation, the aforementioned gates alone are not sufficient [13], and at least one non-Clifford gate is required. One such gate is the T gate,

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{\pi i}{4}} \end{pmatrix}.$$

The Clifford+ T gate set, consisting of the gates $\{H, S, CX, T\}$, is universal [53].

The Z -basis measurement operation M_Z projects a single-qubit quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ onto the Z basis; that is, it returns 0 with the probability $|\alpha|^2$ and returns 1 with the probability $|\beta|^2$. The quantum state becomes $|0\rangle$ when the measurement returns 0, and $|1\rangle$ when it returns 1. We call the return value the *measurement outcome*. The X -basis measurement operation M_X and the Y -basis measurement operation M_Y are defined in the same manner. More generally, we define the

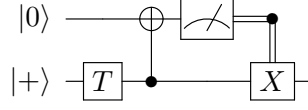


Figure 2.1: Example quantum circuit.

Pauli measurement M_P for a Pauli operator P as the projective measurement with projectors

$$\Pi_m = \frac{I + (-1)^m P}{2},$$

where $m \in \{0, 1\}$ represents the measurement outcome.

A *Pauli rotation* is defined as

$$e^{i\theta P} = \cos \theta I + i \sin \theta P,$$

where P is a Pauli operator and $\theta \in \mathbb{R}$ is the rotation angle. A Pauli rotation $e^{i\theta P}$ is (up to a global phase) a Pauli operator if and only if θ is an integer multiple of $\frac{\pi}{2}$, and a Clifford operator if and only if θ is an integer multiple of $\frac{\pi}{4}$. We refer to $e^{i\theta P}$ as a $\frac{\pi}{8}$ rotation when $\theta \in \{\pm \frac{\pi}{8}\}$ and as a $\frac{\pi}{4}$ rotation when $\theta \in \{\pm \frac{\pi}{4}\}$. In particular, the T gate is a $\frac{\pi}{8}$ rotation, i.e., $T = e^{\frac{i\pi}{8}} e^{\frac{-i\pi Z}{8}}$.

2.1.3 Quantum circuit

A set of qubits together with quantum gates and measurements acting on them forms a *quantum circuit*, representing a quantum computation process. Figure 2.1 illustrates an example of a quantum circuit. Each horizontal line corresponds to a qubit, and time progresses from left to right. In the figure, the top qubit is initialized to $|0\rangle$ and the bottom qubit to $|+\rangle$ at the beginning of the computation. Each box with a symbol, such as the boxed T and X in the figure, represents a gate labeled by its name. A filled control dot connected to a circled plus $\oplus$ denotes a CX gate, with the dot marking the control qubit and the circled plus marking the target qubit. A meter symbol represents an M_Z operation. We also use boxed symbols M_X , M_Y , and M_Z to denote measurement operations. A double line connecting an M_Z operation to an X gate represents *classical control*: the X gate is applied only if the measurement outcome is 1.

2.1.4 Quantum computer hardware

Various hardware platforms have been proposed for realizing quantum computers. The trapped-ion platform offers long coherence times and high gate fidelities but has slow gate speeds and scalability issues. The superconducting platform offers fast gate speeds but suffers from short coherence times and limited connectivity, as interactions are typically limited to neighboring qubits. This restriction is referred to as two-dimensional (2D) *nearest-neighbor connectivity*. Quantum error correction has been experimentally demonstrated on this platform [11]. The neutral-atom platform offers long coherence times and movable qubits, allowing more relaxed connectivity, but operates with

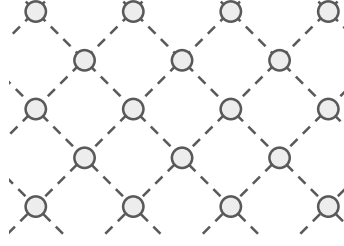


Figure 2.2: Square-grid connectivity. Each circle represents a qubit. A two-qubit gate can be applied to a pair of qubits connected by a dashed line.

slow gate speeds. Experimental realization of quantum error correction on this platform has been reported [12].

In this thesis, we focus on quantum computers powered by superconducting qubits. We additionally assume that qubits are arranged on a square lattice, where neighboring qubits can interact with each other (Figure 2.2). We call this connectivity the *square-grid connectivity*. The square-grid connectivity is among the simplest forms of 2D nearest-neighbor connectivity.

2.2 Quantum error correction

The goal of fault-tolerant quantum computation [9] is to simulate the execution of an ideal (noise-free) quantum circuit using imperfect (noisy) hardware. To this end, quantum error correction (QEC) [10] is used to protect quantum information from noise. Each gate is also implemented in a fault-tolerant manner to protect the computational process. Thus, QEC plays a central role in fault-tolerant quantum computation.

In this section, we first describe the stabilizer formalism [54] in Section 2.2.1. It provides a powerful framework for defining and analyzing a wide range of QEC codes, known as *stabilizer codes*. All QEC codes that appear in this thesis are stabilizer codes. To correct errors using QEC codes, we must first identify the error locations. We describe the process in Section 2.2.2. Next, we define fault-tolerant quantum computation (FTQC) and describe its construction in Section 2.2.3. The rotated surface code [40], introduced in Section 2.2.4, is a promising QEC code for quantum computers powered by superconducting qubits, owing to its high threshold and compatibility with 2D nearest-neighbor connectivity. Lattice surgery [41], described in Section 2.2.5, is a primitive used to implement logical Clifford gates fault-tolerantly for the rotated surface code. Finally, we discuss the complementary gap [55] in Section 2.2.6. It is a method that reduces the logical error probability, at the expense of the success probability, and plays an important role in magic state distillation described later in this thesis.

2.2.1 Stabilizer formalism

For any positive integer n , a *stabilizer* [54] is defined as an Abelian (commutative) subgroup of the Pauli group $\mathcal{P}_n$ that does not contain $-I$. For any stabilizer S , a subset $G = \{g_1, \dots, g_k\} \subset S$ generates S if every element of S can be written as a product of elements of G . We write $S = \langle G \rangle$

or $S = \langle g_1, \dots, g_k \rangle$ and call the elements $g_1, \dots, g_k$ *stabilizer generators*.

A QEC code $\mathcal{C}$ stabilized by a stabilizer S is defined as

$$\mathcal{C} = \{|\psi\rangle \in \mathbb{C}^{2^n} \mid s|\psi\rangle = |\psi\rangle, \forall s \in S\}.$$

The code space $\mathcal{C}$ forms a Hilbert subspace of $\mathbb{C}^{2^n}$. We refer to quantum states in $\mathcal{C}$ as *code states*. If S is generated by independent $n - k$ generators $\{g_1, \dots, g_{n-k}\}$, $\dim \mathcal{C} = 2^k$, and we say that k *logical qubits* are encoded into $\mathcal{C}$ consisting of n *physical qubits*. We call operators acting on logical qubits *logical operators*. A logical operator preserves the code space, i.e., it commutes with all elements of the stabilizer. We attach the symbol “L” to denote logical quantum states, operators, and gates. For example, the three-qubit quantum bit-flip code is stabilized by $S = \langle Z_1 Z_2, Z_2 Z_3 \rangle$. In this case, $n = 3$ and $k = 1$, that is, one logical qubit is encoded into the QEC code. $|0\rangle$ is encoded as $|000\rangle$, i.e., $|0_L\rangle = |000\rangle$. Similarly, $Z_L = Z_1$ and $X_L = X_1 X_2 X_3$. Note that Z_1 and Z_2 have the same action on the code space and hence either can be chosen as Z_L .

A QEC code $\mathcal{C}$ stabilized by a stabilizer S is a *Calderbank-Shor-Steane (CSS)* [56–58] code if and only if S can be generated by X -only generators G_X and Z -only generators G_Z , that is, $S = \langle G_X \cup G_Z \rangle$, where each element of G_X (G_Z) is a tensor product of single-qubit X (Z) operators. All QEC codes that appear in this thesis are CSS codes. Moreover, $\mathcal{C}$ is said to be *self-dual* if it is invariant under the interchange of X and Z operators, that is, if G_Z is obtained from G_X by replacing each X operator with Z on the same set of qubits.

The *weight* of a Pauli operator is defined as the number of qubits that the Pauli operator acts on nontrivially. The *code distance* of $\mathcal{C}$ is the minimum weight of logical Pauli operators that are not equivalent to the logical identity operator. For example, the code distance of the three-qubit quantum bit-flip code is 1, because Z_1 is a nontrivial logical Pauli operator and its weight is 1. Let us denote the code distance as d ; then the QEC code is capable of correcting up to $\lfloor \frac{d-1}{2} \rfloor$ single-qubit errors on the physical qubits. We say that a QEC code is an $[[n, k, d]]$ code when it encodes k logical qubits into n physical qubits and its code distance is d .

In some situations, we want different levels of protection against X and Z errors, and therefore consider the X and Z code distances separately. The X code distance of $\mathcal{C}$ is defined as the minimum weight of logical Pauli operators each of which is (1) a tensor product of X operators and (2) not equivalent to the logical identity operator. The Z code distance is defined in a similar fashion. For example, the X code distance of the three-qubit quantum bit-flip code is 3, whereas its Z code distance is 1.

Steane seven-qubit code

Here we introduce the Steane seven-qubit code [57], a self-dual CSS code that is used later in magic state distillation (Section 2.4). The Steane seven-qubit code is a $[[7, 1, 3]]$ code, defined by the stabilizer

$$S = \langle X_0 X_1 X_4 X_5, X_0 X_2 X_3 X_5, X_0 X_2 X_4 X_6, Z_0 Z_1 Z_4 Z_5, Z_0 Z_2 Z_3 Z_5, Z_0 Z_2 Z_4 Z_6 \rangle.$$

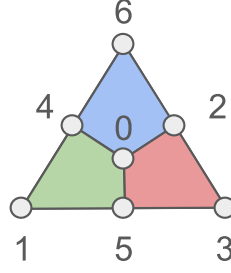


Figure 2.3: Steane seven-qubit code. Each circle represents a physical qubit and its associated number denotes the qubit index. Each colored cell represents a pair of X and Z stabilizer generators.

Figure 2.3 illustrates the code. In the figure, each circle represents a physical qubit and each colored cell represents a pair of X and Z stabilizer generators; for example, the cell spanning 0, 1, 4, 5 represents the pair of $X_0X_1X_4X_5$ and $Z_0Z_1Z_4Z_5$. The tensor product of X (Z) operators acting on physical qubits along a boundary is a logical X (Z) operator, i.e., $X_L = X_1X_4X_6$ and $Z_L = Z_1Z_4Z_6$. Note that any boundary can be chosen for logical operators; for example, $X_1X_3X_5$, $X_1X_4X_6$, and $X_2X_3X_6$ have the same action on the code space.

The Steane seven-qubit code is the smallest 2D color code [39]. 2D color codes admit transversal implementation of Clifford gates (Section 2.2.3), which is useful for magic state distillation, as discussed later in Section 2.4.

2.2.2 Decoding

Quantum error correction involves two procedures: *syndrome extraction* (or *syndrome measurement*) and *decoding*. Syndrome extraction measures stabilizer elements—referred to as *stabilizer checks*—by running quantum circuits (*syndrome extraction circuits*) to obtain error information. A set of such information is called an *error syndrome*. Decoding is a classical computation process that identifies the locations of errors from the error syndrome. Once the decoding process identifies error locations, corrections are applied using quantum gates. The last step can be performed classically using the Pauli feedforward mechanism [59].

Let $|\psi\rangle$ denote a code state of a QEC code $\mathcal{C}$. Let $|\psi'\rangle$ denote the quantum state obtained after noise is applied to $|\psi\rangle$ and error correction is performed. If $|\psi'\rangle$ differs from $|\psi\rangle$, we say that a *logical error* has occurred. The probability that a logical error occurs is called the *logical error probability*.

A related concept is *error detection*. In contrast to error correction, error detection merely detects the existence of an error, and hence no decoding process is needed. Error detection is often used in conjunction with *postselection*, which aborts the whole computation whenever a nontrivial error syndrome is observed. Error detection with postselection generally achieves lower logical error probabilities than error correction for a given QEC code. However, it aborts the computation with nonzero probability and is therefore not scalable.

In our target quantum computer (Section 2.1.4), every gate may cause errors on the qubits it acts on. Note that this also applies to the identity gate: a qubit may suffer an error even when it is merely idling; we refer to this as *idling noise*. As a consequence, syndrome extraction circuits are

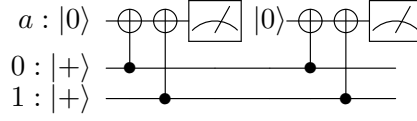


Figure 2.4: An example of a detector.

noisy, and the decoding process must diagnose errors in both the syndrome extraction circuits and the data qubits. The concepts of detectors and detector error models [60, 61] are useful in such circumstances, and we therefore introduce them here.

A *detector* is defined as the parity of measurement outcomes that is deterministic in the absence of errors. For example, Figure 2.4 illustrates two rounds of syndrome extraction. Each syndrome extraction circuit performs an indirect measurement of $Z_0 Z_1$. Although the individual measurement outcomes are non-deterministic, the parity of these outcomes is always 0 in the absence of errors and thus forms a detector. We say that a Pauli error is detected by a detector if the error changes the parity. For example, in Figure 2.4, an X error on qubit 0 occurring between the two rounds is detected by the detector.

A *Tanner graph* [62, 63] is a useful data structure for capturing the error structure of a syndrome extraction circuit. A Tanner graph is a bipartite graph (U, V, E) where U and V are disjoint sets of vertices, and E is the set of edges between U and V . In our context, U represents error events, V represents detectors, and an edge exists between an error and a detector if and only if the detector detects the error. We call a probability assignment to all errors a *noise model*. A *detector error model* is defined as a Tanner graph together with a noise model. A QEC decoder diagnoses the locations of errors, given a detector error model and an error syndrome.

Matching decoders [64] require that each error with nonzero probability be detected by at most two detectors in the detector error model. Such a detector error model is therefore said to be compatible with matching decoders. For example, the detector error model for the typical syndrome extraction circuit of a rotated surface code [40] (Section 2.2.4) is compatible with matching decoders (by decomposing the detector error model into a pair of detector error models for X -only errors and Z -only errors), whereas the detector error model for the typical syndrome extraction circuit of the Steane seven-qubit code [57] (Section 2.2.1) is not.

2.2.3 Fault-tolerant quantum computation

A QEC code protects stored quantum information from noise. In contrast, *fault-tolerant quantum computation* [9] protects the whole computation process from noise. This is achieved by simulating the execution of a noise-free quantum circuit (*logical circuit*) with another quantum circuit built from noisy gates. To this end, qubits in the original circuit are encoded in a QEC code, and each noiseless gate (*logical gate*) in the original circuit is implemented by *fault-tolerant gadgets* built from noisy gates (*physical gates*). Figure 2.5 is an example of this construction. We call a virtual quantum computer that executes a logical circuit using fault-tolerant quantum computation a *fault-tolerant quantum computer*. When there is no risk of confusion, we use the abbreviation

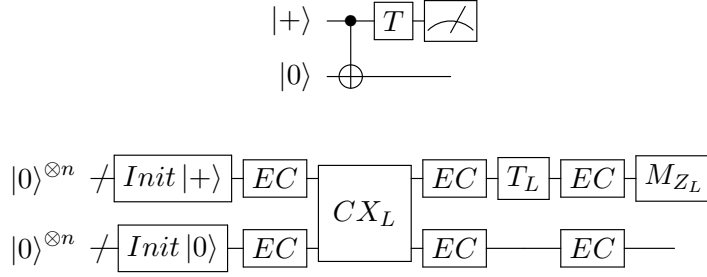


Figure 2.5: An example of fault-tolerant quantum computation. Top: A logical circuit. Bottom: A physical circuit simulating the logical circuit. Each horizontal line represents a collection of n physical qubits, in which one logical qubit is encoded. “Init” denotes initialization of a logical qubit, “EC” stands for error correction.

FTQC to refer to both fault-tolerant quantum computation and fault-tolerant quantum computer.

The threshold theorem [20, 65, 66] promises that the above construction allows efficient simulation of a noise-free circuit with arbitrary precision if the noise strength is below a threshold. The threshold value depends on the choice of QEC codes and fault-tolerant gadgets, as well as on the noise model. For example, Aliferis *et al.* [67] proved the threshold theorem for the independent stochastic noise model using concatenation of a distance-3 QEC code, and calculated a lower bound on the threshold value, 2.73×10^{-5} .

To guarantee fault-tolerance, it is crucial that each logical gate is implemented in a fault-tolerant manner, that is, a fault-tolerant gadget does not spread errors so that the subsequent error correction can correct them. *Transversal implementation*, which is a logical gate implementation with qubit-wise application of physical gates, does not spread errors and is therefore inherently fault-tolerant. However, there is no universal logical gate set each of which is implemented transversally [68].

2.2.4 Surface code

The surface code [30, 69] is a QEC code that has a high threshold and is compatible with the square-grid connectivity, making it a promising QEC code for quantum computers powered by superconducting qubits. The rotated surface code [40] is a variant of the surface code that reduces the number of physical qubits required. In this thesis, we use the rotated surface code, and we describe it here.

The rotated surface code with distance d encodes one logical qubit into d^2 physical data qubits arranged on a $d \times d$ square lattice. We also need an ancillary qubit for each stabilizer check, and hence $2d^2 - 1$ qubits are required in total. The X and Z stabilizer checks are placed in a checkerboard pattern. We call the set of these $2d^2 - 1$ physical qubits a *surface code patch*, or simply *patch*. Figure 2.6 depicts a distance-5 surface code patch. In the figure, the tensor product of X operators acting on data qubits placed on a vertical boundary is a logical X operator. The tensor product of Z operators acting on data qubits placed on a horizontal boundary is a logical Z operator. We call a surface code boundary that hosts a logical X (Z) operator an X (Z) boundary. Hence,

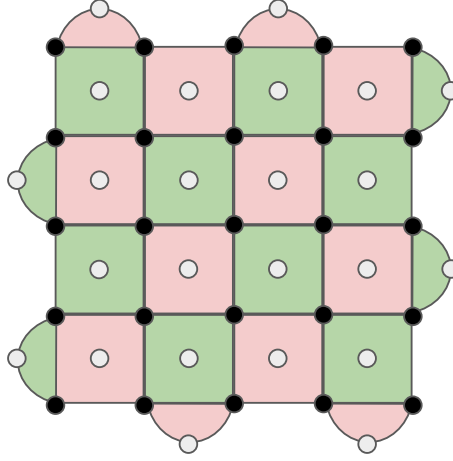


Figure 2.6: A distance-5 rotated surface code patch. A red (green) square represents a four-weight X (Z) stabilizer check. A red (green) semicircle represents a two-weight X (Z) stabilizer check. A black circle represents a physical data qubit. A gray circle represents a measurement qubit for syndrome extraction.

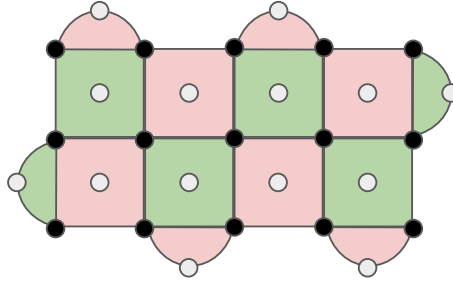


Figure 2.7: A 3×5 surface code patch.

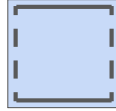


Figure 2.8: A simplified surface code patch illustration. Solid and dashed lines represent Z and X boundaries, respectively. These lines may be omitted when not relevant to computation.

in the figure, a vertical boundary is an X boundary and a horizontal boundary is a Z boundary. This explains why the code distance corresponds to the linear length of the square. Intuitively, d single-qubit X (Z) errors occurring along an X (Z) boundary cause a logical X (Z) error.

We also consider a surface code patch with different X distance d_X and Z distance d_Z . In this case, the patch is rectangle-shaped, reflecting the asymmetric code distances. We call such a patch a $d_X \times d_Z$ surface code patch. Figure 2.7 depicts a 3×5 surface code patch. In some cases, we are interested in surface code patches rather than individual physical qubits. Then we depict a surface code patch as a simple square (or rectangle if it has different X and Z distances), as illustrated in Figure 2.8.

More flexible patch constructions can also be considered. For example, one can create a patch that ensures a code distance d but is larger than a $d \times d$ patch. Moreover, a patch can host multiple

logical qubits by carefully placing stabilizer checks. These constructions are useful because they allow flexible associations between logical operators and the boundaries used by lattice surgery (Section 2.2.5). Such constructions are used extensively in sequential Pauli-based computation [26] (Section 2.3.2); as illustrated later in Figure 2.20, a patch whose size is equivalent to two $d \times d$ patches and that hosts two logical qubits is used in the “fast block” in Ref. [26]. In this layout, boundary segments on different sides of the patch are assigned to the logical operators of each qubit, allowing the Pauli operators of both qubits to be directly measured via lattice surgery without requiring additional patch movement or rotation. This enables compact storage while still allowing efficient implementation of multi-qubit Pauli measurements.

The rotated surface code is a CSS code, and its stabilizer can be generated by X -only generators and Z -only generators. This is illustrated in Figure 2.6, where red (green) cells represent X (Z) stabilizer checks. The X stabilizer checks detect Z errors, whereas the Z stabilizer checks detect X errors. Thus, the detector error model of the rotated surface code can be decomposed into two detector error models for X errors and Z errors, each of which is compatible with matching decoders. Therefore, the minimum-weight perfect matching (MWPM) decoder [70], which maps the decoding problem to a graph matching problem and solves it using Edmonds’ algorithm [71, 72], can be used for decoding. Specifically, we run the MWPM decoder with the detector error model for X errors, then run the decoder with the detector error model for Z errors. A single Y error introduces correlations between the two detector error models because it produces detection events in both models. These correlations are incorporated by reweighting [73] the detector error model used in the second decoding run based on the result of the first run.

We now discuss the fault-tolerant implementation of logical gates for the rotated surface code. We begin by clarifying timing assumptions. We use the time required to perform a round of syndrome extraction (*cycle time*) as the time unit. Classical branching incurs an additional latency, referred to as the *reaction time*. In chapter 4 and Section 2.4.4, we assume a reaction time of 10 cycles.¹ Elsewhere in this thesis, the reaction time is assumed to be negligible compared to a single cycle and is therefore ignored. This difference arises because we compare performance results with prior studies that adopt different assumptions.

Preparing $|0_L\rangle$ ($|+_L\rangle$) can be done fault-tolerantly by initializing each physical qubit to $|0\rangle$ ($|+\rangle$) and performing d rounds of syndrome extraction, and therefore it requires d cycles [26]. Note that this state preparation operation can often be performed in parallel with subsequent operations. For example, creating a Bell pair requires initializing two patches to $|+\rangle$ and performing a ZZ measurement through lattice surgery. All of these can be performed in parallel, and hence the Bell pair initialization requires d cycles in total. The rotated surface code admits transversal implementation of Pauli gates and X and Z -basis measurement. It also admits a transversal implementation of the CX gate. However, this is not compatible with 2D nearest-neighbor connectivity, and we use lattice surgery for the fault-tolerant implementation of Clifford gates including CX . We need magic state distillation (Section 2.4) for the fault-tolerant implementation of a non-Clifford gate.

¹In chapter 4, we use the term “round”, following preceding studies. Although closely related to “cycle”, the two notions are defined slightly differently; see Section 4.3.1 for details.

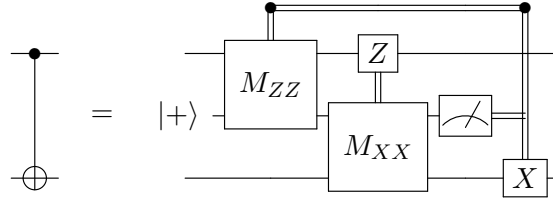


Figure 2.9: CX implemented by two multi-qubit Pauli measurements.

2.2.5 Lattice surgery

Lattice surgery [41] enables logical multi-qubit Pauli measurements in a manner compatible with 2D nearest-neighbor connectivity. Logical multi-qubit Pauli measurements are used, for example, to implement a logical CX gate, as shown in Figure 2.9. In the figure, M_{ZZ} and M_{XX} denote two-qubit Pauli measurements implemented by lattice surgery.

Lattice surgery consists of two steps—namely the merge and split steps—as illustrated in Figure 2.10. In Figure 2.10 (left), two distance-3 patches A and B are placed. Let us assume that A is stabilized by $S_A = \langle G_A \rangle$ and B is stabilized by $S_B = \langle G_B \rangle$. We also name physical qubits $a-i$. $Z_{L_A} = Z_{abc}$ is a logical Z operator of A and $Z_{L_B} = Z_{ghi}$ is a logical Z operator of B . We can regard all physical data qubits except d, e , and f as forming a large QEC code in which two logical qubits are encoded. The large QEC code is stabilized by $\langle G_A \cup G_B \rangle$ and its distance is 3.

Next, we initialize qubits d, e , and f to $|+\rangle$ and place stabilizer checks Z_{abde} , X_{bcef} , Z_{cf} , Z_{dg} , $X_{deg h}$, and Z_{efhi} (Figure 2.10 (middle)). The large QEC code (including physical qubits $d-f$) is stabilized by

$$\langle G_A \cup G_B - \{X_{bc}, X_{gh}\} \cup \{(-1)^x Z_{abde}, X_{bcef}, (-1)^y Z_{cf}, (-1)^z Z_{dg}, X_{deg h}, (-1)^u Z_{efhi}\} \rangle.$$

Here, the factors $(-1)^x$, $(-1)^y$, $(-1)^z$, and $(-1)^u$ indicate that the outcomes of these stabilizer measurements are non-deterministic. x, y, z , and $u \in \{0, 1\}$ are the measurement outcomes.

In the split step, we measure physical qubits $d-f$ in the X basis (Figure 2.10 (right)). Then the large QEC code (again ignoring physical qubits $d-f$) is stabilized by

$$\langle G_A \cup G_B \cup \{(-1)^{x+y+z+u} Z_{abde} Z_{cf} Z_{dg} Z_{efhi}\} \rangle = \langle G_A \cup G_B \cup \{(-1)^{x+y+z+u} Z_{abc} Z_{ghi}\} \rangle,$$

meaning that the quantum state is the result of performing a logical $Z_{L_A} Z_{L_B}$ measurement, and the measurement outcome is $x \oplus y \oplus z \oplus u \in \{0, 1\}$.

This operation can be applied to patches with arbitrary distance d . A single measurement error on the merged boundary (in the above example, x, y, z , and u) during the merge step can flip the measurement outcome $(x \oplus y \oplus z \oplus u)$. In order to deal with these measurement errors, each syndrome measurement in the merge step needs to be performed d times. During the sequence, the code distance is preserved; that is, the operation is fault-tolerant. The entire process consists of single-qubit initialization and measurement, along with local syndrome measurements, and is

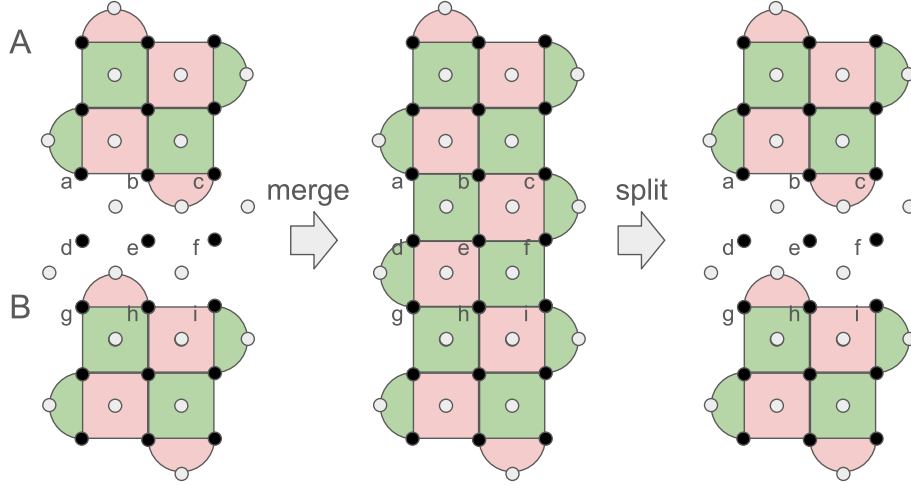


Figure 2.10: Performing lattice surgery on two distance-3 patches along the Z boundaries.

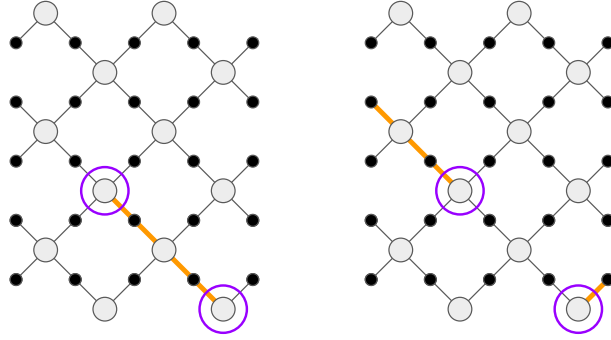


Figure 2.11: Tanner graphs of the distance-5 rotated surface code (for Z errors only). Gray, black, and purple circles indicate detectors, error locations, and detectors that detect Z errors, respectively. The left and right panels show matching results that are complementary to each other, with matching paths colored orange.

therefore compatible with 2D nearest-neighbor connectivity.

A logical XX measurement can be implemented in the same manner. Logical measurements with more flexible axis choices can be implemented, though with more involved implementations [74]. These allow fault-tolerant implementations of Clifford gates. Lattice surgery can also be performed between two different QEC codes [42], and among many patches [26].

2.2.6 Complementary gap

The *complementary gap* [55] quantifies the decoder's confidence in its decoding result given an error syndrome, and it can be computed using the MWPM decoder. In this subsection, for simplicity, we consider only Z errors. We also assume that syndrome extraction circuits are noise-free, and each syndrome measurement forms a detector, which is sometimes called the *code capacity noise model*. Figure 2.11 shows the Tanner graphs of the distance-5 rotated surface code under these assumptions. Each X stabilizer check corresponds to a detector (gray circle), and each data qubit corresponds to an error location (black circle).

The MWPM decoder solves the matching problem to identify error locations. Figure 2.11 shows two matching solutions, with matching paths colored orange. Both are valid matching results, and they are complementary to each other in the sense that the union of the matching paths forms a logical Z chain. If the two solutions are equally likely, the decoder’s confidence in the decoding result is low.

The complementary gap is a measure used to quantify the difference in the likelihood of two logical classes. To compute its value, the MWPM decoder is run twice with modified detector error models [75], once for a logical class and once for its complementary class. The complementary gap is defined as the absolute difference between the weights of the two matching results. Later in this thesis, we use this value for the final postselection of magic state distillation protocols. Specifically, a distillation attempt is aborted when the complementary gap falls below the given threshold.

2.3 Compilation schemes for FTQC

A fault-tolerant quantum computer is a complex system composed of multiple layers. In this thesis, we model it as a sequence of layers. In Section 2.3.1, we introduce the hierarchical model and define the concept of an FTQC compilation scheme, while Section 2.3.2 discusses sequential Pauli-based computation (SPC) [26], an FTQC compilation scheme. Note that for universal computation, we need magic state distillation, described in Section 2.4. This section appears before Section 2.4 because some magic state distillation protocols in that section utilize SPC.

2.3.1 FTQC compilation

We model an FTQC as a sequence of layers ordered from high level to low level. This hierarchical model follows the model proposed in Ref. [33]. The hierarchical model consists of the following layers, as illustrated in Figure 2.12:

1. **Quantum high-level programming language:** A high-level programming language used for describing quantum algorithms, such as Q# [76] and Quipper [77].
2. **Quantum IR:** An intermediate representation (IR) such as QIR [78] that lacks high-level language features.
3. **Quantum ISA:** A set of fault-tolerant gadgets that implement quantum IR instructions.
4. **QEC code:** As discussed in Section 2.2.4, we use the rotated surface code in this thesis.
5. **Hardware:** A quantum device accompanied by classical computers, on which a QEC code is implemented.

Although any translation process from a higher layer to a lower layer can be called compilation, in this thesis we use the term specifically for a translation from the quantum IR layer to the QEC code layer. We also refer to *transpilation* as a translation process from the quantum IR layer to the quantum ISA layer. We define a *compilation scheme* as a quantum ISA specification accompanied

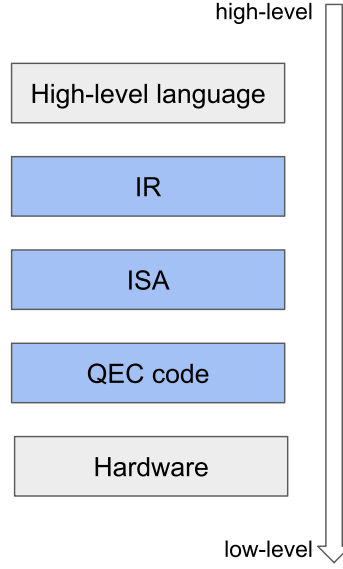


Figure 2.12: The hierarchical model of FTQC, consisting of multiple layers.

by a compilation algorithm. Sequential Pauli-based computation (Section 2.3.2) is an example of a compilation scheme. In this thesis, we assume that a quantum IR consists of the following operations:

1. Single-qubit $|0\rangle$ initialization,
2. Single-qubit Z measurement,
3. H , S , and CX Clifford gates,
4. $e^{i\theta Z}$, where $\theta \in [0, 2\pi)$, and
5. Classical control.

Classical control includes branching and looping. Although classical control is an important component of quantum computation, we omit it in this thesis to simplify the discussion. Note that magic state distillation is probabilistic and may require repeated attempts, introducing loop structures in the execution of non-Clifford gates. The compilation schemes considered in this thesis provide dedicated support for such magic state preparation; therefore, general-purpose classical control instructions are not required for execution of non-Clifford gates.

Having specified the quantum IR, we now describe the main features of FTQC compilation. Among the various FTQC compilation features, we focus on four: *routing*, *mapping*, *scheduling*, and *gate synthesis*. Because our target quantum computer has nearest-neighbor connectivity, performing a two-qubit logical gate requires a path between the patches corresponding to the two logical qubits. Suppose we wish to perform a ZZ measurement on two logical qubits. If the two qubits are adjacent, we can perform lattice surgery (Figure 2.13, q_1 and q_2). Otherwise, an intermediate path of patches must be created between them to perform a ZZ measurement (Figure 2.13, q_3 and q_4). There may be multiple possible paths between a pair of qubits, and the compiler selects one. This process is referred to as *routing*.

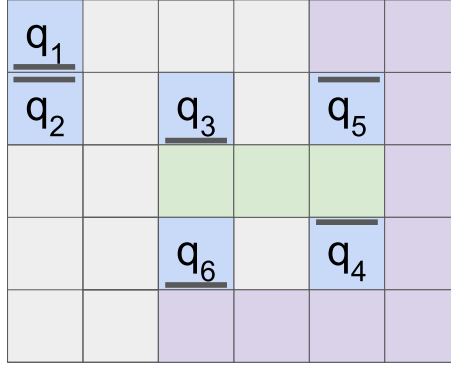


Figure 2.13: Routing examples. Lattice surgery can be performed directly on q_1 and q_2 because they are adjacent and their logical Z operators are aligned. The green region represents a path required to perform $M_{Z_3Z_4}$. The purple region depicts a path required to perform $M_{Z_5Z_6}$ without conflicting with the green path. Each square is a surface code patch; see Figure 2.8.

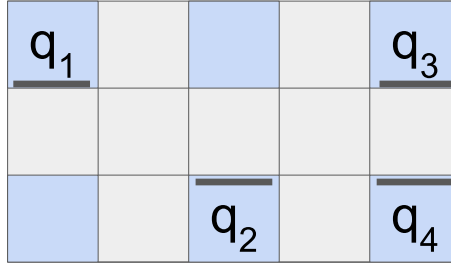


Figure 2.14: An example of scheduling. Because $M_{Z_1Z_4}$ and $M_{Z_2Z_3}$ cannot be executed in parallel, the compiler determines their execution order.

The computational cost of performing a two-qubit gate depends on the positions of the target qubits. Thus, the assignment of a surface code patch to a logical qubit affects the total execution time. We refer to this association process as *mapping*.

The compiler also assigns a time slot to each operation, ensuring that the semantics of the input program are preserved. For example, in Figure 2.14, $M_{Z_1Z_4}$ and $M_{Z_2Z_3}$ cannot be performed simultaneously because M_{ZZ} requires a path that connects the logical Z operators of the target qubits, and no two paths can simultaneously connect q_1 to q_4 and q_2 to q_3 without intersecting. Since $M_{Z_1Z_4}$ and $M_{Z_2Z_3}$ commute, the compiler can determine their execution order. We call this process *scheduling*.

Some algorithms require arbitrary-angle Z rotations of the form $e^{i\theta Z}$, where $\theta \in [0, 2\pi)$. *Gate synthesis* is the process of approximating an arbitrary-angle rotation using a circuit consisting of Clifford, T , initialization, and measurement operations. In this thesis, we use the mixed diagonal algorithm [79], which approximates an arbitrary-angle rotation by decomposing it into a sequence of Clifford and T gates. The sequence contains approximately 1.5δ T gates, where $2^{-\delta}$ denotes the precision of the rotation angle. Although other algorithms, such as the mixed fallback algorithm [79], achieve a lower T count, we use the mixed diagonal algorithm for its simpler circuit structure.

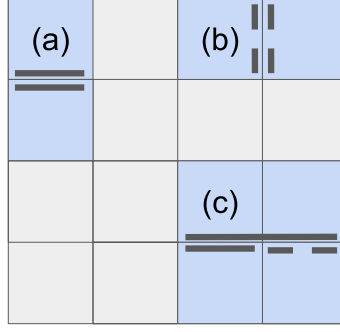


Figure 2.15: Examples of multi-qubit Pauli measurements implemented via lattice surgery: (a) M_{ZZ} , (b) M_{XX} , and (c) M_{ZY} . Each square is a surface code patch; see Figure 2.8.

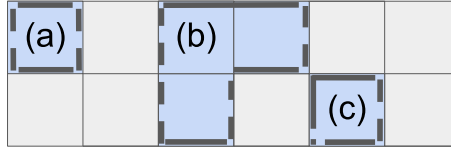


Figure 2.16: Qubit initialization examples. (a) Single-patch initialization. (b) Multi-qubit initialization. (c) Single-patch initialization with a shortened X boundary.

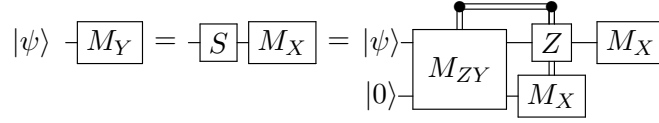


Figure 2.17: Y measurement implementation using M_{ZY} .

2.3.2 Sequential Pauli-based computation

In this subsection, we describe sequential Pauli-based computation, a compilation scheme introduced by Litinski [26]. We first define the quantum ISA and its implementation, and then describe the algorithm that translates the quantum IR into the quantum ISA.

Quantum ISA and its implementation

The quantum ISA for sequential Pauli-based computation consists of the following operations:

1. Single-qubit $|0\rangle$ initialization,
2. Single-qubit Pauli measurement,
3. Multi-qubit Pauli measurement, and
4. $\frac{\pi}{8}$ Pauli rotation $e^{\pm \frac{\pi i}{8} P}$ for some Pauli operator P .

Sequential Pauli-based computation utilizes lattice surgery to implement a multi-qubit Pauli measurement (Figure 2.15). Figure 2.15 (c) uses a twist defect to implement a multi-qubit Pauli measurement involving Y .

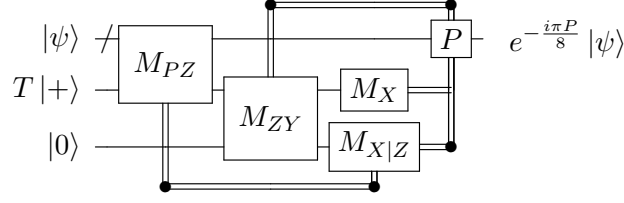


Figure 2.18: $\frac{\pi}{8}$ rotation implementation. P is a Pauli operator. M_{PZ} and M_{ZY} run in parallel. The measurement axis of $M_{X|Z}$ depends on the measurement outcome of M_{PZ} .

$$\begin{aligned}
 \text{---} \boxed{P} \text{---} \boxed{P'} \text{---} &= \begin{cases} \text{---} \boxed{P'} \text{---} \boxed{P} \text{---} & (PP' = P'P) \\ \text{---} \boxed{iPP'} \text{---} \boxed{P} \text{---} & (PP' = -P'P) \end{cases} & \text{---} \boxed{P} \text{---} \boxed{P'} \text{---} &= \begin{cases} \text{---} \boxed{P'} \text{---} & (PP' = P'P) \\ \text{---} \boxed{iPP'} \text{---} & (PP' = -P'P) \end{cases}
 \end{aligned}$$

Figure 2.19: Gate translation rules for SPC. P and P' denote Pauli operators. An orange, green, and blue box labeled P represents a $\frac{\pi}{4}$ rotation $e^{-\frac{i\pi}{4}P}$, a $\frac{\pi}{8}$ rotation $e^{-\frac{i\pi}{8}P}$, and a Pauli measurement M_P , respectively.

$|0\rangle$ and $|+\rangle$ initializations require a single patch and d cycles (Figure 2.16 (a)). Multi-qubit initializations are also possible with d cycles (Figure 2.16 (b)). For Z (X) eigenstates, the Z (X) boundaries can be shortened (Figure 2.16 (c)).

Single-qubit Z and X measurements are implemented via transversal Z and X measurements. A single-qubit Y measurement is implemented with $|0\rangle$ initialization, lattice surgery, and a transversal Z measurement (Figure 2.17). Multi-qubit Pauli measurements are implemented with lattice surgery. Since transversal Z and X measurements require only one cycle—much shorter than d cycles—we neglect this time cost in the following discussion. With this approximation, single-qubit Z and X measurements each require 0 cycles, whereas single-qubit Y and multi-qubit Pauli measurements each require d cycles.

A $\frac{\pi}{8}$ Pauli rotation $e^{-\frac{\pi i}{8}P}$ is implemented using multi-qubit Pauli measurements with a magic state (Section 2.4), as shown in Figure 2.18. This operation also requires d cycles.

Transpilation

A quantum program expressed in the quantum IR is translated into the quantum ISA through the following steps. First, each arbitrary-angle Z rotation is decomposed into a sequence of Clifford and T gates using gate synthesis (Section 2.3.1). Next, each Clifford gate is decomposed into a sequence of $\frac{\pi}{4}$ rotations. For example, $H = e^{\frac{i\pi}{4}Z} e^{\frac{i\pi}{4}X} e^{\frac{i\pi}{4}Z}$ and $CX_{12} = e^{\frac{i\pi}{4}Z_1 X_2} e^{-\frac{i\pi}{4}Z_1} e^{-\frac{i\pi}{4}X_2}$. In the final step, $\frac{\pi}{4}$ rotations are moved to the end of the circuit using the rules depicted in Figure 2.19. The $\frac{\pi}{4}$ rotations at the end of the circuit can be omitted since they do not affect the measurement outcomes. The resulting circuit consists of single-qubit initialization, Pauli measurements, and $\frac{\pi}{8}$ rotation gates. Thus, the resulting circuit is compatible with the quantum ISA described in Section 2.3.2.

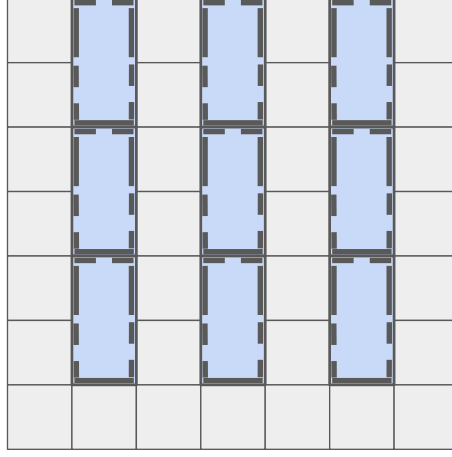


Figure 2.20: Layout for sequential Pauli-based computation with $N = 18$. Magic state distillation factories are not included. Each blue rectangle hosts two data qubits.

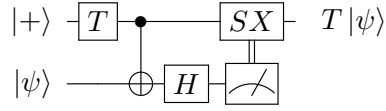


Figure 2.21: Gate teleportation circuit for applying a T gate.

We next discuss scheduling, mapping, and routing in this compilation scheme. Through the above translation, a single-qubit $\frac{\pi}{8}$ Z rotation becomes a $\frac{\pi}{8}$ P rotation, where P is a multi-qubit Pauli operator. As a result, performing multiple $\frac{\pi}{8}$ rotations in parallel is challenging, even when they commute. Hence, sequential Pauli-based computation executes each rotation and measurement sequentially, making scheduling trivial. This also simplifies mapping and routing, since supporting sequential gate execution is straightforward. We use the layout depicted in [Figure 2.20](#), which requires $2N + \sqrt{8N} + 1$ patches, each a $d \times d$ patch—excluding magic state distillation factories—where N is the number of data qubits and d is the code distance.

2.4 Magic state distillation

As discussed in [Section 2.2.3](#), in FTQC, each logical gate must be implemented in a fault-tolerant manner. While fault-tolerant implementations of Clifford gates are relatively inexpensive, Clifford gates alone are not sufficient for universal quantum computation [13]. Implementing a non-Clifford gate such as the T gate requires a special quantum state, the *magic state*, which is obtained through a process known as *magic state distillation* [14]. Recursive application of magic state distillation—called *multi-level distillation*—enables the preparation of magic states with arbitrarily high fidelity. In general, different non-Clifford gates require different types of magic states. Such gates include the T and CCZ gates, which can be implemented using the corresponding magic states $|T\rangle := T|+\rangle$ and $|CCZ\rangle := CCZ|+\rangle^{\otimes 3}$, respectively.

As an example, consider applying the T gate to a single-qubit quantum state $|\psi\rangle$. This can be

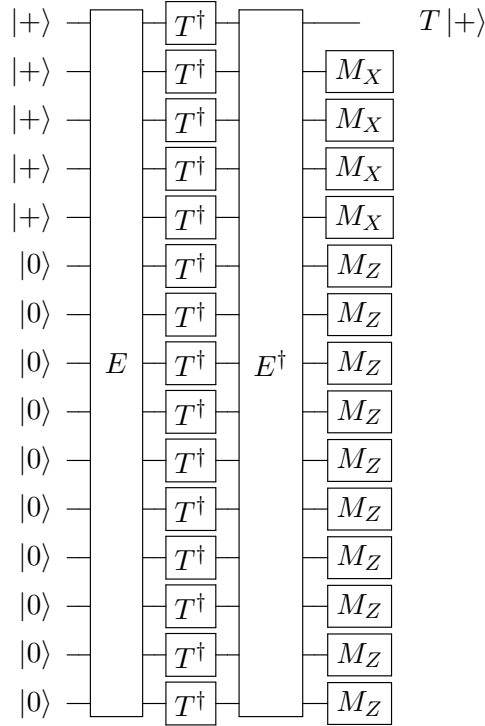


Figure 2.22: The 15-to-1 magic state distillation protocol. E is an encoding circuit to the $[[15, 1, 3]]$ code.

done using gate teleportation and a magic state $T |+\rangle$, as illustrated in Figure 2.21. Using magic states is preferable to applying the T gate to the input state $|\psi\rangle$ directly, because postselection can be used to prepare high-fidelity magic states without risking the destruction of the input state.

Bravyi *et al.* [14] proposed the 15-to-1 distillation protocol, which exploits T 's transversality on the $[[15, 1, 3]]$ code to generate $T |+\rangle$. As illustrated in Figure 2.22, the protocol encodes $|+\rangle$ into the $[[15, 1, 3]]$ code, performs T transversally, and then decodes the encoded $T |+\rangle$. The final measurements detect up to two errors arising from non-Clifford gates. An event in which an error is detected by the final measurements is referred to as a *distillation failure*. In addition to implementing a T gate, the $T |+\rangle$ state can be used as a resource for distilling other magic states. In particular, the $|CCZ\rangle$ state, which enables implementation of the CCZ gate, can be distilled from eight $T |+\rangle$ states [24]. Earlier, before the term “magic state distillation” was introduced, Knill *et al.* [20] proposed a protocol that utilizes H 's transversality on the Steane code [57] (Section 2.2.1) and generates $|H\rangle$, the $+1$ eigenstate of H . As illustrated in Figure 2.23, this protocol encodes $|H\rangle$ into the Steane code, performs a Hadamard test of H , and then decodes the encoded $|H\rangle$. The measurements detect up to two errors arising from non-Clifford gates and the $|H\rangle$ initialization.

Notably, these studies formulated magic state distillation as a quantum algorithm built from ideal Clifford gates and noisy inputs. Noisy non-Clifford gates in the distillation circuits ($T^\dagger$ in Bravyi's protocol and controlled- H in Knill's protocol) are implemented with ideal Clifford gates and noisy quantum states, for example, using gate teleportation. This formulation was followed by

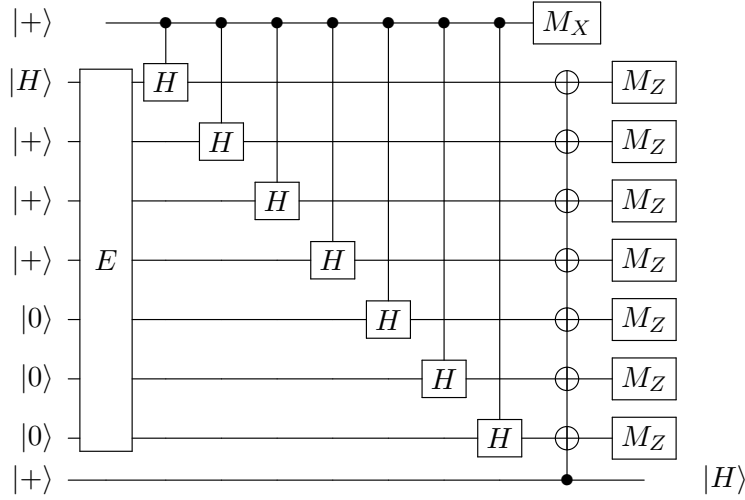


Figure 2.23: Knill’s distillation protocol. E is an encoding circuit to the Steane seven-qubit code.

subsequent studies [21, 22, 24–26]. Running such algorithms on an FTQC requires protecting each Clifford gate against noise using QEC, incurring a significant cost.

Goto [23] proposed a technique that we call *physical-level distillation*. In contrast to the above formulation, the distillation circuit operates with noisy Clifford gates. Specifically, the circuit detects any single gate error, thereby achieving a logical error probability of $\mathcal{O}(p_{\text{phys}}^2)$, where p_{phys} is the physical error probability. To this end, the distillation protocol consists of a Knill-style Hadamard test with a flag qubit. Its physical-gate implementation significantly reduces the distillation overhead.

Goto’s protocol requires all-to-all connectivity. Chamberland *et al.* [28] and Itogawa *et al.* [29] proposed physical-level distillation protocols that work with planar connectivity. The latter, known as zero-level distillation, operates on the simpler square-grid connectivity and yields a magic state encoded in a surface code [30, 31], achieving a logical error probability of 10^{-4} for a realistic $p_{\text{phys}} = 10^{-3}$ setting.

The rest of this section is structured as follows. In [Section 2.4.1](#), we discuss the 15-to-1 distillation protocol, with its implementation using sequential Pauli-based computation (SPC) [26] ([Section 2.3.2](#)). In [Section 2.4.2](#), we discuss the two-level 15-to-1 distillation implementation using SPC. In [Section 2.4.3](#) and [Section 2.4.4](#), we describe zero-level distillation and magic state cultivation [38], which are physical-level distillation protocols that work with the square-grid connectivity.

2.4.1 15-to-1 distillation

The 15-to-1 magic state distillation protocol [14] yields a magic state, given fifteen copies of low-fidelity magic states as input. The implementation of the 15-to-1 protocol in Refs. [26, 27] employs a sequence of $\frac{\pi}{8}$ Pauli rotations on five logical qubits [80], which is executed efficiently with sequential Pauli-based computation ([Section 2.3.2](#)). Let Γ be a set of Pauli operators defined

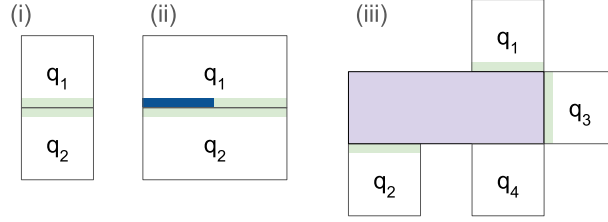


Figure 2.24: Examples of block diagrams depicting (i) M_{ZZ} , (ii) M_{YZ} , and (iii) M_{ZZZ} . Each rectangle except the purple one is a surface code patch (see Section 2.2.4). Blue and green lines on the patch boundaries represent logical operators involved in lattice surgery, and the purple rectangle connects patches.

as follows:

$$\Gamma = \left\{ \prod_{i \in s} Z_i \mid s \subset \{1, 2, 3, 4, 5\}, |s| \text{ is odd} \right\}.$$

The product of $\frac{\pi}{8}$ Pauli rotations across Γ is the identity operator acting on the five qubits:

$$\prod_{P \in \Gamma} e^{\frac{i\pi}{8} P} = I.$$

Note that all the operators in Γ commute and so do the $\frac{\pi}{8}$ Pauli rotations across Γ . Because $Z_1 \in \Gamma$, separating the factor corresponding to Z_1 from the full product gives

$$\prod_{P \in \Gamma, P \neq Z_1} e^{\frac{i\pi}{8} P} = e^{-\frac{i\pi}{8} Z_1}.$$

Applying the product of rotations to $|+\rangle^{\otimes 5}$ yields

$$\left(\prod_{P \in \Gamma, P \neq Z_1} e^{\frac{i\pi}{8} P} \right) |+\rangle^{\otimes 5} = e^{-\frac{i\pi}{8} Z} |+\rangle \otimes |+\rangle^{\otimes 4},$$

and therefore we obtain $e^{-\frac{i\pi}{8} Z} |+\rangle$, which is $|T\rangle$. Because the rotations act on the five qubits nontrivially, we can detect errors in the rotations by measuring the last four qubits in the X basis.

In this subsection, we illustrate quantum computation processes using two graphical representations. One is a conventional quantum circuit, such as Figure 2.25 (left). The other is a block diagram, which conveys the patch layout and lattice surgery information, similar to the figures in Ref. [26]. For example, Figure 2.24 (i) depicts M_{ZZ} , where the green lines between the patches signify logical Z operators on the patches; (ii) illustrates M_{YZ} , with the blue line indicating a logical X operator; and (iii) represents M_{ZZZ} , where the purple rectangle connects patches involved in the measurement (q_4 is not involved in the measurement). Each of these operations requires d cycles, where d is the code distance (see Section 2.2.5 and Section 2.3.2).

The single-level 15-to-1 implementation uses *faulty $T^\dagger$ measurement*—a faulty $T^\dagger$ operator followed by an X measurement—to perform a $\frac{\pi}{8} P$ rotation, where P is a Pauli operator, as illustrated in Figure 2.25 (left). We denote such a rotation as an orange cell connected to the target

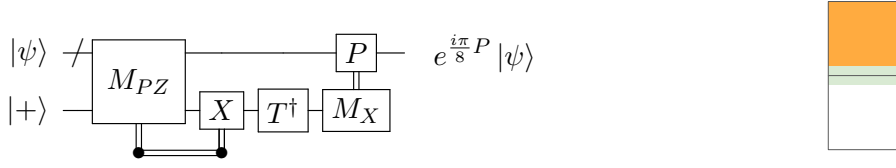


Figure 2.25: Circuit for a $\frac{\pi}{8} P$ rotation with a faulty $T^\dagger$ measurement (left) and its block diagram representation (right, $P = Z$).

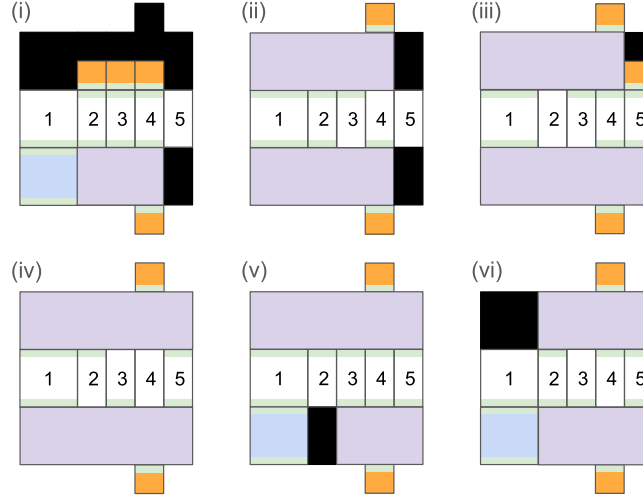


Figure 2.26: Single-level 15-to-1 protocol implementation consisting of six rounds. Numbered cells represent data qubits. Blue cells represent the magic state which is accessible to the outside of the distillation factory. Black cells represent unused qubits.

qubit in block diagrams, as depicted in Figure 2.25 (right). The operation's error probability is comparable to the physical error probability. The distillation process, which consists of six rounds, is depicted in Figure 2.26. For example, in the first round, Z_2 , Z_3 , Z_4 , and $Z_2Z_3Z_4$ rotations are performed. Since each round consists of $\frac{\pi}{8}$ rotations performed simultaneously, it completes in the time required to perform one $\frac{\pi}{8}$ rotation. For multi-qubit Pauli rotations, the involved qubits and the qubits on which the faulty $T^\dagger$ gate is performed need to be connected, similar to Figure 2.24 (iii). The purple regions in Figure 2.26 serve this purpose. We refer to these as *ancilla regions*. There are at most two ancilla regions for each round, namely the top and bottom ancilla regions. At the end of the final round, qubits 2–5 are measured in the X basis, and the measurement results are checked to detect errors. When an error is detected, we need to discard the quantum state at qubit 1; that is, the distillation attempt fails.

To assess the logical error probability of a distilled magic state, it is essential to evaluate errors introduced by each operation within the distillation process. We begin by considering idle errors occurring on the data qubits. The distillation protocol consists of $\frac{\pi}{8}$ rotations, with each rotation's axis commuting with Z operators, allowing Z errors occurring on qubits 2–5 to be detected by the final X measurements. In contrast, Z errors on qubit 1 and X errors across all the data qubits may elude detection, and hence they need to be suppressed more strongly. As shown in Figure 2.27, this is achieved using two distinct code distances—namely d_X and d_Z , where $d_X \geq d_Z$ —to protect the

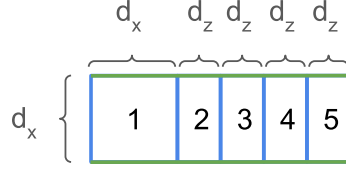


Figure 2.27: Data-qubit patch dimensions. Logical Z (X) operators on the boundaries are colored green (blue).

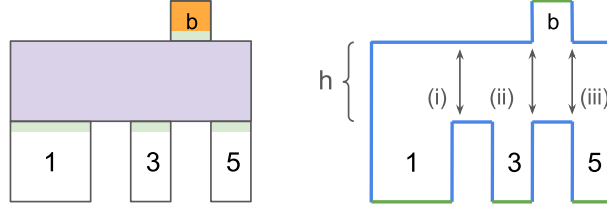


Figure 2.28: $\frac{\pi}{8}$ $Z_1 Z_3 Z_5$ rotation (left) and logical operators on the boundary during the rotation (right).

data qubits against idle errors.

Next, we consider errors caused by faulty $T^\dagger$ measurements. A $\frac{\pi}{8}$ P rotation translates a Pauli error from its faulty $T^\dagger$ measurement into a P rotation, which is detectable by the final X measurements, similarly to Z errors on qubits 2–5. The error probability of the Pauli measurement M_{PZ} in a faulty $T^\dagger$ measurement (Figure 2.25) is determined by the number of cycles spent for the measurement, which we call d_m . Given this observation, we set $d_m = d_Z$, and hence, each round completes in d_Z cycles.

Idle errors on separate data qubits are covered by the above discussion, but idle errors during $\frac{\pi}{8}$ rotations are not. Consider, for instance, performing a $\frac{\pi}{8}$ $Z_1 Z_3 Z_5$ rotation, as illustrated in Figure 2.28 (left). During the rotation, the data qubits involved in the rotation and the logical qubit on which the faulty $T^\dagger$ gate is performed (b in the figure) are merged via the ancilla region (the purple rectangle in the figure) by lattice surgery. Figure 2.28 (right) illustrates the logical operators on the boundary of the merged qubits. Green (blue) lines represent logical Z (X) operators. As shown in the figure, logical Z operators on the boundary are distant from each other, which means the lengths of logical X operators are long. Hence, the error probability of idle X errors during the rotation is very small. In contrast, Z errors can be problematic when h in the figure is short. Figure 2.28 (i), (ii), and (iii) represent Z errors spanning X operators on the boundary. (i) is equivalent to Z_1 whereas (ii) is equivalent to $Z_1 Z_3$ and (iii) is equivalent to $Z_1 Z_3 Z_b \simeq Z_5$. In this case, (i) is the most severe because a Z_1 error is not detectable by the final X measurements. Therefore, we set $h = d_X$ to suppress the error with the same level of effectiveness as for idle Z errors on qubit 1. We denote this distillation implementation $(15\text{-to-}1)_{d_X, d_Z}$, where d_X and d_Z are parameters.²

²The implementation in Ref. [27] contains another distance parameter, d_m . For simplicity, in this thesis, we always set $d_m = d_Z$.

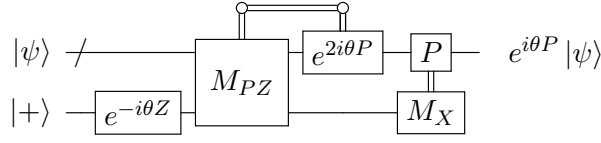


Figure 2.29: Circuit implementing $e^{i\theta P}$ with state injection.

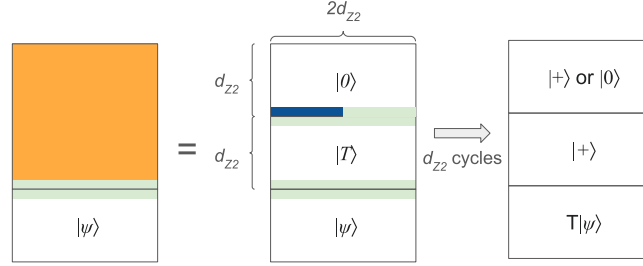


Figure 2.30: Block diagram of an auto-correction circuit.

2.4.2 (15-to-1) $\times$ (15-to-1) distillation

In this subsection, we describe the two-level 15-to-1 distillation implementation using SPC (Section 2.3.2). The $(15\text{-to-}1)_{d_{X1}, d_{Z1}} \times (15\text{-to-}1)_{d_{X2}, d_{Z2}}$ protocol implementation, which utilizes $(15\text{-to-}1)_{d_{X1}, d_{Z1}}$ to generate first-level magic states, is similar to the single-level 15-to-1 protocol implementation but differs in several ways. The first difference is that the first-level distillation process operates more slowly than a second-level $\frac{\pi}{8}$ rotation. The implementation places n_{L1} first-level distillation factories and establishes pipelines to offset the speed difference, for an even integer n_{L1} . This also means that running more than two $\frac{\pi}{8}$ rotations within a round, which the single-level implementation does in the first and third rounds (Figure 2.26), is difficult. Hence, the two-level distillation process requires 7.5 rounds to complete, which means that an eight-round distillation is followed by a seven-round distillation, rather than six rounds of the single-level protocol. We also need to transfer magic states from the first-level distillation factories to perform second-level $\frac{\pi}{8}$ rotations; this transfer incurs additional spatial overhead.

Another difference is that we cannot use a faulty $T^\dagger$ measurement at the second level. Instead, we use state injection to perform a $\frac{\pi}{8}$ rotation, as illustrated in Figure 2.29. This method requires a $\frac{\pi}{4}$ rotation with the probability of 1/2, potentially incurring a 50% additional temporal overhead to the total distillation process, if unaddressed. To eliminate this temporal overhead for the $\frac{\pi}{4}$ rotation, we use an auto-correction circuit. While it uses more qubits (four $d_{Z2} \times d_{Z2}$ patches instead of one), we can perform a $\frac{\pi}{8}$ rotation with d_{Z2} cycles, as depicted in Figure 2.30. For consistency in block diagrams, this is also denoted as an orange cell in block diagrams, because state injection and faulty $T^\dagger$ measurement serve analogous purposes and they are not used at the same time. An auto-correction block can perform a $\frac{\pi}{8}$ rotation within d_{Z2} cycles, but it requires another d_{Z2} cycles to transfer a magic state into it. Hence, to keep performing two $\frac{\pi}{8}$ rotations for each round, we need four auto-correction blocks in total.

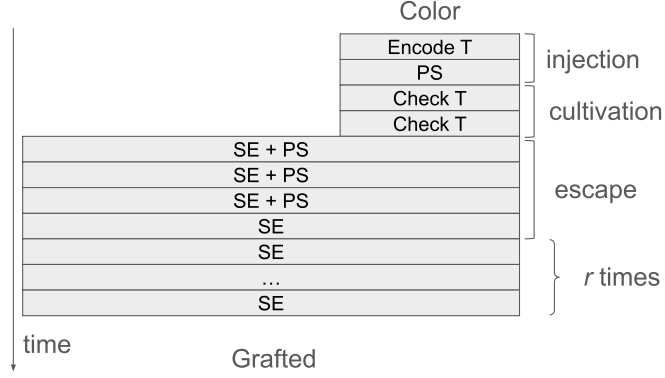


Figure 2.31: Magic state cultivation [38] for $d_{\text{color}} = 3$. “Encode T” denotes the non-fault-tolerant encoding of a magic state into the distance-3 2D color code. Two consecutive “Check T” represent the application of two Hadamard tests. “PS” denotes postselection, and “SE” denotes syndrome extraction.

2.4.3 Zero-level distillation

Zero-level distillation [29] is a physical-level distillation protocol that operates on the square-grid connectivity and yields $|H\rangle = e^{-\frac{i\pi}{8}Y} |+\rangle$, which is the $+1$ eigenstate of the Hadamard gate H . Similar to the preceding physical-level distillation protocols [23, 28], this protocol performs distillation using a Hadamard test of H , relying on the fact that the Steane seven-qubit code [57] admits transversal implementation of H .

The distillation protocol consists of three stages: (1) encoding $|H\rangle$ non-fault-tolerantly into the Steane code, (2) performing a Hadamard test of H to distill the magic state, and (3) teleporting the magic state into the rotated surface code [40] (Section 2.2.4) using lattice surgery [41, 42] (Section 2.2.5). The original zero-level distillation yields $|H\rangle = e^{-\frac{i\pi}{8}Y} |+\rangle$, but it can also yield $|T\rangle = e^{-\frac{i\pi}{8}Z} |+\rangle$ by changing the rotation axis.

Zero-level distillation requires 40 physical qubits and 25 physical depths. The 40 physical qubits fit into a single 5×5 surface code patch. A single round of syndrome measurement on the rotated surface code consists of up to four CNOTs and two measurements. Hence, one cycle corresponds to six physical depths, and zero-level distillation completes in five cycles.

The logical error probability of a magic state generated by zero-level distillation is approximately $100p_{\text{phys}}^2$, where p_{phys} is the physical error probability. The failure probability is approximately 30% when $p_{\text{phys}} = 10^{-3}$ and 5% when $p_{\text{phys}} = 10^{-4}$.

2.4.4 Magic state cultivation

Magic state cultivation [38] is an efficient magic state distillation protocol. It consists of three stages: injection, cultivation, and escape. In the injection stage, a magic state is injected non-fault-tolerantly into the distance-3 2D color code [39], which is also known as the Steane seven-qubit code [57] (Section 2.2.1). In the cultivation stage, the fault distance of the magic state is increased to d_{color} . Here, we define the *fault distance* of a circuit or a state as the minimum number of errors required to cause a logical error without being detected by any detector. In the escape stage, the color

code is transformed into the grafted code, another QEC code compatible with matching decoders (Section 2.2.2). Finally, we compute the complementary gap [55] (Section 2.2.6) and perform postselection based on its value. We perform r rounds of syndrome extraction while waiting for the QEC decoder to compute the complementary gap. Figure 2.31 depicts the protocol for $d_{\text{color}} = 3$.

Injection stage

In the injection stage, similar to the preceding physical-level distillation protocols [23, 29], a magic state is injected non-fault-tolerantly into the distance-3 2D color code, also known as the Steane seven-qubit code [57]. The original magic state cultivation proposal provides three injection methods: unitary injection, teleport injection, and Bell injection. In this thesis, we use unitary injection, which is an injection circuit composed of unitary gates, throughout because it has the best performance of the three. After the magic state is injected, we perform one round of syndrome extraction followed by postselection; if the error syndrome is nontrivial, we reject the distillation attempt. At the end of this stage, a magic state with fault distance 1 is encoded in the distance-3 2D color code.

Cultivation stage

In the cultivation stage, the fault distance of the magic state is increased. Conceptually, this is done by performing Hadamard tests, as in preceding proposals using physical-level distillation [23, 28, 29]. This relies on two facts: (1) the magic state $|T\rangle = T|+\rangle$ is the +1 eigenstate of $H_{XY} = (X+Y)/\sqrt{2}$, and (2) 2D color codes admit transversal implementation of H_{XY} . Magic state cultivation employs an efficient implementation of a pair of Hadamard tests (corresponding to the two “Check T” boxes in Figure 2.31) that is well-suited to the square-grid connectivity.

When $d_{\text{color}} = 3$, the cultivation stage ends here. With a larger d_{color} , the 2D color code is expanded and the code distance increases by 2. After performing syndrome extraction and postselection for code stabilization, another pair of Hadamard tests is performed to increase the fault distance. This procedure is repeated until the fault distance reaches d_{color} .

At the end of this stage, a magic state with fault distance d_{color} is encoded in the 2D color code of distance d_{color} . Similarly to the injection stage, in this stage, any nontrivial measurement outcome leads to rejection of the distillation attempt.

Escape stage

In the escape stage, the 2D color code of distance d_{color} is transformed into the grafted code of distance d_{grafted} , another QEC code compatible with matching decoders. The transformation involves three steps: (1) creating a combination of a surface code and a color code, (2) stabilizing the code by measuring its stabilizer checks several times, and (3) making the color code region matchable by dropping some stabilizer checks and decomposing others into two-body stabilizer checks. During the stabilization step, we manually perform postselection for some detectors on the color code region and remove them from the detector error model (Section 2.2.2) used by the QEC decoder to keep the detector error model compatible with matching decoders.

Because the grafted code is compatible with matching decoders, we can compute the complementary gap and perform postselection based on it. The grafted code of distance d is similar to the rotated surface code of distance d but has a higher logical error probability. For example, Gidney *et al.* [38] report that the logical error probability of the grafted code of distance 15 is comparable to that of the rotated surface code of distance 11.

2.5 Example quantum circuits for benchmarking

In this section, we describe several quantum circuits that are used later in this thesis for performance evaluation. When evaluating the performance of FTQC systems, it is important to use concrete quantum circuits, as they expose practical trade-offs that are not captured by abstract metrics alone. In this thesis, we focus on efficient preparation and consumption of magic states. Specifically, we are interested in quantum algorithms that allow parallel execution.

In [Section 2.5.1](#), we describe random circuit sampling, which is primarily used for benchmarking quantum computers, for example, quantum computational supremacy demonstrations [8]. Other algorithms are more directly relevant to practical applications. In [Section 2.5.2](#), we introduce the SELECT oracle [81], which is an important oracle used in the phase estimation algorithm [82] and the quantum singular value transformation algorithm [83]. We also describe its distributed variant [84]. In [Section 2.5.3](#), we describe a Hamiltonian simulation of the 2D transverse-field Ising model using Trotterization.

2.5.1 Random circuit sampling

Random circuit sampling samples from the probability distribution of randomly generated quantum circuits. Due to the average-case hardness of the task on classical computers [85] and the relative ease of experimental realization, it serves as a benchmark for demonstrating quantum computational supremacy [8]. Random circuit sampling is structured in layers, each of which consists of two steps. In the first step, a CZ gate is applied to each pair of neighboring qubits. In the second step, each qubit undergoes a randomly chosen single-qubit gate from $\{S, H, T\}$.

2.5.2 SELECT oracle

The SELECT oracle [81] is an important oracle used in the phase estimation algorithm [82] and the quantum singular value transformation algorithm [83], both of which have a wide variety of applications, including quantum chemistry [86]:

$$\text{SELECT} := \sum_{\ell=0}^{L-1} |\ell\rangle\langle\ell| \otimes H_{\ell},$$

where each ℓ is an integer expressed with $\log L$ qubits (*index qubits*) and each H_{ℓ} is a Pauli operator that operates on N qubits (*Pauli target qubits*). For simplicity, we assume that L is a power of two. We call L the length of the SELECT oracle. We can naively implement the oracle with $\mathcal{O}(L \log L)$

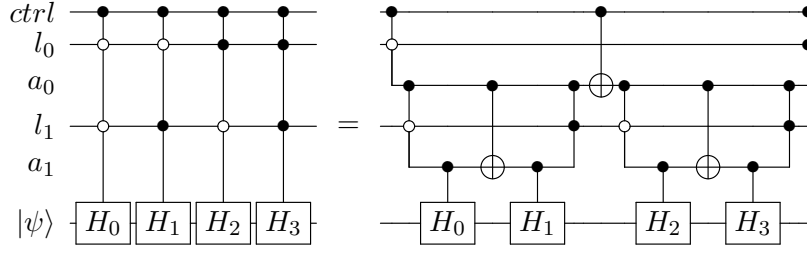


Figure 2.32: SELECT circuit with $L = 4$.

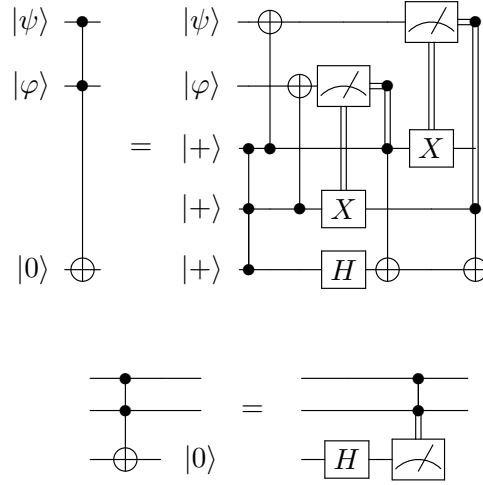


Figure 2.33: Toffoli gates used in the SELECT circuit.

Toffoli gates and $\log L$ ancilla qubits. Babbush *et al.* [81] proposed an implementation with $\mathcal{O}(L)$ Toffoli gates (Figure 2.32).

In the optimized SELECT circuit (Figure 2.32), we use two types of Toffoli gates: a Toffoli gate applied to a target qubit initialized to $|0\rangle$, and its inverse. Figure 2.33 shows their implementations. The former is implemented with gate teleportation and a CCZ resource state $|CCZ\rangle := CCZ |+\rangle^{\otimes 3}$. The latter is implemented without magic states.

Yoshioka *et al.* [84] proposed a distributed SELECT circuit (Dist-SELECT), which shortens the circuit execution time by parallelizing the execution. Let M be a power of two less than L . Dist-SELECT with parallelism M consists of parallel-running M sub-circuits each of which runs a SELECT operation with length $\frac{L}{M}$, as depicted in Figure 2.34. This increases the number of qubits because of the additional index and ancilla qubits, but as long as M is sufficiently smaller than N , the additional spatial overhead is relatively small. At the expense of the additional qubits, each SELECT operation has a shorter length, which may lead to a shorter execution time. Because all sub-circuits share the Pauli target qubits, actual parallelism depends on how many controlled Pauli operators we can run in parallel on the Pauli target qubits. This depends on (1) each Pauli operator H_ℓ determined by the problem itself, and (2) the routing efficiency determined by the quantum

computer architecture and the compiler.

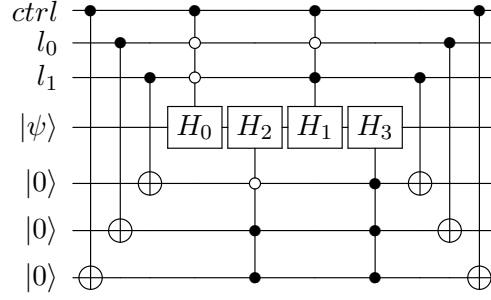


Figure 2.34: Dist-SELECT circuit with $L = 4$, $M = 2$, consisting of two parallel-running sub-circuits.

2.5.3 Hamiltonian simulation with Trotterization

Hamiltonian simulation is one of the most promising applications of quantum computing. As a prototypical example, we employ a quantum circuit to simulate the 2D transverse-field Ising model using Trotterization, following Refs. [33, 87]. We consider the time evolution e^{-iHt} of a Hamiltonian

$$H = -JA + gB = -J \sum_{\langle j,k \rangle} Z_j Z_k + g \sum_j X_j,$$

where J and g are constants.

To approximate e^{-iHt} , we split it into T steps, where T is a positive integer. Recall that $e^{-iHt} = (e^{-iHt/T})^T$. Each $e^{-iHt/T}$ is approximated using the following fourth-order Trotterization approximation:

$$\begin{aligned} U_2(\Delta) &= e^{-iB\Delta/2} e^{-iA\Delta} e^{-iB\Delta/2}, \\ U_4(\Delta) &= (U_2(\gamma\Delta))^2 U_2((1-4\gamma)\Delta) (U_2(\gamma\Delta))^2, \\ \gamma &= (4 - 4^{\frac{1}{3}})^{-1}. \end{aligned}$$

Here, $U_4(\Delta)$ approximates $e^{-iH\Delta}$ up to an error of $\mathcal{O}(\Delta^5)$. T is chosen so that the total error remains within an acceptable range. The decomposition yields a sequence of $5T$ A exponentials and $5T + 1$ B exponentials, which can be directly translated into the quantum IR (Section 2.3.1).

Chapter 3

Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States

Magic state distillation plays an important role in universal fault-tolerant quantum computing, and its overhead is one of the major obstacles to realizing fault-tolerant quantum computers. Hence, many studies have been conducted to reduce this overhead. Among these efforts, Litinski has provided a concrete assessment of resource-efficient distillation protocol implementations on the rotated surface code. On the other hand, Itogawa *et al.* have recently proposed zero-level distillation, a distillation protocol offering very small spatial and temporal overhead required to generate relatively low-fidelity magic states. While zero-level distillation offers favorable spatial and temporal overhead, it cannot directly generate high-fidelity magic states since it only reduces the logical error probability of the magic state quadratically. In this chapter, we evaluate the spatial and temporal overhead of two-level distillation implementations generating relatively high-fidelity magic states, including ones incorporating zero-level distillation. To this end, we introduce (0+1)-level distillation, a two-level distillation protocol that combines zero-level distillation and the 15-to-1 distillation protocol. We refine the second-level 15-to-1 implementation in it to capitalize on the small footprint of zero-level distillation. Under conditions of a physical error probability of $p_{\text{phys}} = 10^{-4}$ (10^{-3}) and targeting an error probability for the magic state within $[5 \times 10^{-17}, 10^{-11}]$ ($[5 \times 10^{-11}, 10^{-8}]$), (0+1)-level distillation reduces the spacetime overhead by more than 63% (61%) compared to the (15-to-1) $\times$ (15-to-1) protocol and more than 43% (44%) compared to the (15-to-1) $\times$ (20-to-4) protocol, offering a substantial efficiency gain over the traditional protocols. This chapter is based on “**Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States,**” Hirano, Itogawa, Fujii, *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 843–853 [49].

3.1 Introduction

Quantum computers promise to solve practically important problems that are also classically intractable, such as integer factoring [3], quantum chemistry [6], and solving large-scale linear systems [7]. Currently available quantum computers, also known as noisy intermediate-scale

quantum computers (NISQ) [88], are too vulnerable to hardware errors to run complex quantum programs. In order to realize a theoretically proven quantum speedup, we need a fault-tolerant quantum computer (FTQC) [9].

FTQC is built on top of quantum error-correcting (QEC) codes to suppress noise, and computation is protected from errors by encoding logical qubits using many physical qubits, thereby adding spatial and temporal overhead. Reducing this overhead, which is the main obstacle to realizing FTQC, is one of the most important issues in this field. Consequently, various QEC codes are being explored and FTQC architectures are being designed [89]. Among these, the surface codes [30, 69] (Section 2.2.4) are among the most promising QEC codes because of their relatively high error threshold with nearest-neighbor connectivity, and we therefore focus on them in this chapter.

Implementing Clifford gates on the surface codes in a fault-tolerant manner can be done relatively efficiently by using lattice surgery [26, 41] (Section 2.2.5). However, Clifford gates are not enough to form a universal gate set with which we can run an arbitrary quantum algorithm; a non-Clifford gate is essentially required. Implementing a non-Clifford gate such as the T gate requires a special quantum state, a *magic state*, which is obtained through a process called *magic state distillation* [14] (Section 2.4). The spatial and temporal overhead of magic state distillation is one of the major bottlenecks of FTQC, especially when many T gates are performed simultaneously, and hence many studies have been conducted to reduce the overhead [24–27, 29, 90].

One primary reason why magic state distillation protocols require large spatial and temporal overhead is that they assume ideal Clifford gates, and thus require logical gates for their implementation. To mitigate this issue, Litinski [27] has proposed resource-efficient implementations of magic state distillation protocols (Section 2.4.1, Section 2.4.2) on the rotated surface code, accompanied by a detailed performance assessment. These implementations reduce the overhead by carefully relaxing the ideality condition of Clifford gates to use fewer physical qubits per logical qubit. Recently, Itogawa *et al.* [29] have proposed zero-level distillation (Section 2.4.3), another efficient implementation of a magic state distillation protocol. It runs distillation fault-tolerantly on physical qubits rather than logical qubits by carefully designing the error-detection circuit, which essentially reduces the number of physical qubits required for distillation. While zero-level distillation offers favorable spatial and temporal overhead, it cannot directly generate high-fidelity magic states since it only reduces the logical error probability of the magic state quadratically. Therefore, it is still not well understood how it contributes to reducing the spatial and temporal overhead for achieving a T gate of a given target fidelity and parallelism.

In this chapter, we evaluate the spatial and temporal overhead of two-level distillation implementations generating relatively high-fidelity magic states, including those incorporating zero-level distillation, to answer the question. To this end, we introduce (0+1)-level distillation, a two-level distillation protocol that combines zero-level distillation and the 15-to-1 distillation protocol. We refine the second-level 15-to-1 implementation in it to capitalize on the small footprint of zero-level distillation. We also address the drawback of zero-level distillation, its relatively higher failure probability. To assess the performance of these distillation implementations, we conduct two performance analyses. One analysis assesses the spatial and temporal overhead of magic state distillation. The other analysis estimates the resources required for a Hamiltonian simulation,

aiming to assess the impact of different distillation protocols on the overall performance of quantum computation.

The first analysis reveals that, under conditions of a physical error probability of $p_{\text{phys}} = 10^{-4}$ (10^{-3}) and targeting an error probability for the magic state within $[5 \times 10^{-17}, 10^{-11}]$ ($[5 \times 10^{-11}, 10^{-8}]$), (0+1)-level distillation reduces the spacetime overhead by more than 63% (61%) compared to the two-level 15-to-1 implementation and more than 43% (44%) compared to the two-level (15-to-1) $\times$ (20-to-4) implementation, offering a substantial efficiency gain over the traditional protocols. The second analysis reveals that, with $p_{\text{phys}} = 10^{-4}$ (10^{-3}), adopting (0+1)-level distillation reduces the total spatial cost by 47% (41%) compared to using the (15-to-1) $\times$ (15-to-1) protocol, and by 44% (33%) compared to the (15-to-1) $\times$ (20-to-4) protocol, suggesting that the impact on the overall performance is substantial.

The rest of this chapter is organized as follows. In [Section 3.2](#), we detail our proposal, (0+1)-level distillation. In [Section 3.3](#), we conduct a comprehensive performance evaluation of (0+1)-level distillation. Finally, [Section 3.4](#) concludes the chapter with a summary of our findings.

3.2 (0+1)-level distillation

This section introduces (0+1)-level distillation, a two-level distillation protocol that combines zero-level distillation and the 15-to-1 distillation protocol, and details its implementation. The implementation builds upon the $(15\text{-to-1})_{d_{X1}, d_{Z1}} \times (15\text{-to-1})_{d_{X2}, d_{Z2}}$ implementation described in [Section 2.4.2](#). Throughout this section, we use the graphical representations introduced in [Section 2.4.1](#). [Section 3.2.1](#) describes the integration of zero-level distillation with the 15-to-1 distillation protocol. Following this, we outline strategies to refine the second-level 15-to-1 distillation implementation for further efficiency ([Section 3.2.2](#) and [Section 3.2.3](#)). Finally, in [Section 3.2.4](#), we combine these strategies and describe the (0+1)-level distillation implementation. Throughout this section, we assume that $d_{X2} \geq d_{Z2} \geq 5$ to simplify our analysis. With this assumption, zero-level distillation is performed in one $d_{Z2} \times d_{Z2}$ patch (spatially) and in d_{Z2} cycles (temporally).

3.2.1 Zero-level distillation integration

Zero-level distillation provides significant reductions in spatial and temporal overhead. By employing it to generate first-level magic states, we significantly reduce the spatial overhead associated with the two-level distillation. With the assumption described above, zero-level distillation is performed in one $d_{Z2} \times d_{Z2}$ patch (spatially) and in d_{Z2} cycles (temporally). Thus, zero-level distillation can be performed within the auto-correction block, which eliminates the spatial cost for first-level distillation factories and qubits necessary to transfer the first-level magic states. [Figure 3.1](#) visually presents this integration, showcasing both the distillation and $\frac{\pi}{8}$ rotation processes, where a zero-level distillation block is represented as a gray cell.

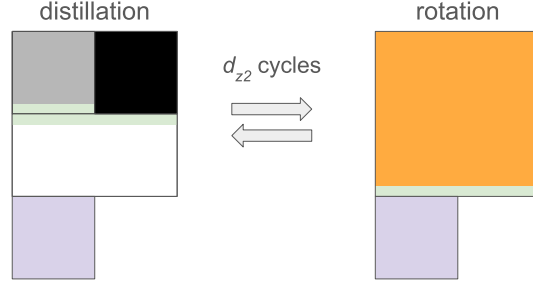


Figure 3.1: Zero-level distillation in an auto-correction block. The gray area represents zero-level distillation. The black area represents unused qubits.

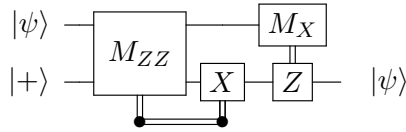


Figure 3.2: Teleportation circuit.

3.2.2 Teleportation

When the target state is $|+\rangle$, instead of performing a $\frac{\pi}{8}$ rotation on the target state, we can use quantum teleportation (Figure 3.2). Because teleportation does not require the $\frac{\pi}{4}$ fix-up rotation required by state injection, we do not need an auto-correction block and hence it is more efficient. Specifically, an auto-correction block uses four patches (spatially) and $2d_{Z2}$ cycles (temporally), whereas teleportation uses one patch (spatially) and d_{Z2} cycles (temporally).

However, the efficiency offered by teleportation comes with caveats. One is that, as stated above, teleportation can be used only when the target state is $|+\rangle$. This also means that when using teleportation, the target qubit must not be involved in other rotations. For instance, in the first round of the single-level 15-to-1 distillation (Figure 2.26), rotations Z_2 , Z_3 , Z_4 , and $Z_2Z_3Z_4$ are performed simultaneously. This is not possible with teleportation.

Next, the desired state is $T^\dagger |+\rangle = e^{\frac{i\pi}{8}Z} |+\rangle$, rather than $|T\rangle = e^{-\frac{i\pi}{8}Z} |+\rangle$. This is not problematic because zero-level distillation can generate the desired state, $e^{\frac{i\pi}{8}Z} |+\rangle$.

Finally, zero-level distillation has a relatively high failure probability. Faulty $T^\dagger$ measurement always succeeds, although the operation itself is faulty. Zero-level distillation, on the other hand, fails with a nonzero probability. For example, with a physical error probability $p_{\text{phys}} = 10^{-3}$, it fails with a probability of 30%. Thus, when four teleportation operations are performed simultaneously to set qubits 2–5, at least one teleportation fails with a probability of 76%, resulting in a longer distillation process.

In Section 2.4.1, X errors on qubits 2–5 are suppressed more strongly than Z errors on them. One may wonder if errors on the magic state lead to problems, especially when there is an X error and $XT^\dagger |+\rangle$ is teleported to the target qubit. This is not problematic, though, because $XT^\dagger |+\rangle = e^{-\frac{i\pi}{4}Z} T^\dagger |+\rangle$ (up to global phase), and the error is detectable by the final X measurements.

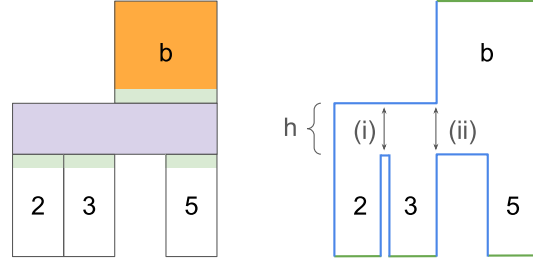


Figure 3.3: $\frac{\pi}{8} Z_2 Z_3 Z_5$ rotation (left) and the logical operators on the boundary during the rotation (right).

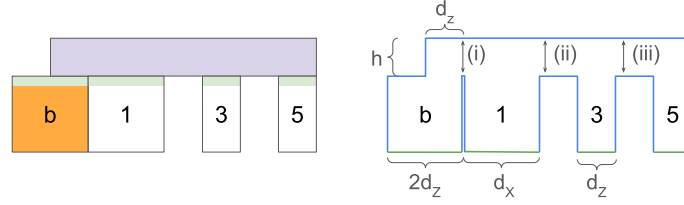


Figure 3.4: $\frac{\pi}{8} Z_1 Z_3 Z_5$ rotation (left) and the logical operators on the boundary during the rotation (right).

3.2.3 Shortening the height of the ancilla region

In [Section 2.4.1](#), the height of the ancilla regions of the 15-to-1 implementation, which is h in [Figure 2.28](#), is set to d_X . This setting is based on the observation that some Z error chains of length h are equivalent to a Z error on qubit 1, which is an undetectable error. However, not all Z error chains pose the same level of threat; for instance, Z error chains on [Figure 2.28](#) (ii) and (iii) are equivalent to $Z_1 Z_3$ and Z_5 , respectively, both of which are detectable. In this subsection, we refine the layout of the second-level distillation process to avoid undetectable Z error chains of length h , thereby allowing h to be reduced, for further spatial efficiency.

When qubit 1 is not involved in a $\frac{\pi}{8}$ rotation, no error chains on the ancilla region contain an error equivalent to a Z_1 error, thereby making all Z errors detectable. [Figure 3.3](#) illustrates one such example. Z error chains on [Figure 3.3](#) (i) and (ii) are equivalent to Z_2 and $Z_2 Z_3$, respectively. Consequently, under this condition, h can be reduced.

Even when qubit 1 is involved in a $\frac{\pi}{8}$ rotation, undetectable Z errors can be circumvented by strategically placing auto-correction blocks. The interpretation of a Z error chain that connects the top and bottom sides of the ancilla region is determined by the arrangement of data qubits and the auto-correction block. A Z error chain on [Figure 2.28](#) (i) is equivalent to Z_1 because it is topologically equivalent to a chain that connects the left and right edges of qubit 1. This situation can be avoided by placing the auto-correction block immediately to the left of qubit 1. [Figure 3.4](#) depicts the $\frac{\pi}{8} Z_1 Z_3 Z_5$ rotation with this layout. Z error chains on [Figure 3.4](#) (i), (ii), and (iii) are equivalent to $Z_1 Z_3 Z_5$, $Z_3 Z_5$, and Z_5 , respectively, all of which are detectable.

It should be noted, however, that not all detectable errors are equally harmful. The 15-to-1 distillation protocol, which performs 15 rotations, can detect up to two arbitrary rotation errors, contributing to its error suppression capability, characterized by $O(p^3)$. However, certain idle

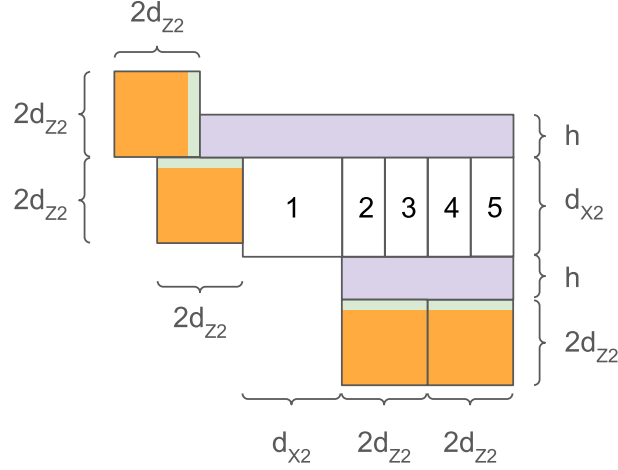


Figure 3.5: (0+1)-level distillation layout. Numbered cells represent data qubits. Purple cells depict the ancilla regions. Orange cells with green boundaries represent qubits used for zero-level distillation and auto-correction.

errors occurring during a $\frac{\pi}{8}$ rotation compromise this property. For instance, a Z error chain on **Figure 3.3** (ii) is equivalent to a Z_2Z_3 error. While this error itself is detectable, the combination of Z_2Z_3 and $Z_1Z_2Z_3$ errors becomes undetectable, given that $Z_2Z_3 \cdot Z_1Z_2Z_3 = Z_1$. Hence, a Z_2Z_3 error is more harmful than other errors, such as Z_4 . Therefore, when prioritizing magic state fidelity over distillation overhead, it is advisable to set h larger than d_{Z_2} .

3.2.4 (0+1)-level distillation

In this subsection, we outline the implementation of (0+1)-level distillation, which has distance parameters (d_{X_2}, d_{Z_2}, h) . It is designed to reduce the spacetime overhead of the conventional (15-to-1) $\times$ (15-to-1) distillation protocol by integrating the strategies we discussed in previous subsections. **Figure 3.5** depicts the layout of the distillation circuit. The top ancilla region handles rotations involving qubit 1, while the bottom ancilla region handles those that do not. As explored in **Section 3.2.3**, the parameter h , representing the height of both ancilla regions, can effectively be reduced to below d_{X_2} , thereby improving spatial efficiency.

Figure 3.6 illustrates the distillation process, which consists of seven rounds. For example, in the first round, two $\frac{\pi}{8}$ Pauli rotations—namely, a $Z_1Z_3Z_4$ rotation and a Z_4 rotation—and one teleportation operation targeting qubit 5 are performed. Given that the distillation process consists of seven rounds, the active and inactive auto-correction blocks alternate roles in the subsequent distillation; for example, the right-bottom auto-correction block is active in even-numbered rounds in the figure, whereas it will be active in odd-numbered rounds in the next distillation. The process illustrated in the figure is designed to seamlessly accommodate this role alternation; for instance, the Z_4 rotation can be performed using the right-bottom auto-correction block because the bottom ancilla region connects qubit 4 to the block. At the end of the final round, qubits 2–5 are measured and the measurement results are checked to detect errors, while qubit 1 is reset. Consequently, establishing entanglement between qubit 1 and an external logical qubit beforehand is essential for

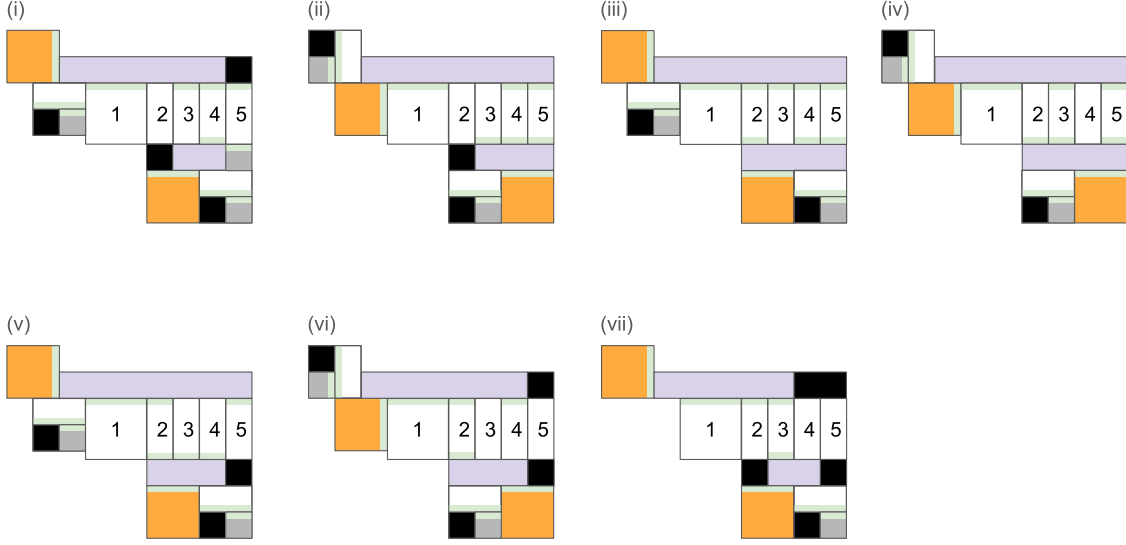


Figure 3.6: (0+1)-level distillation process consisting of seven rounds. Numbered cells represent data qubits. Purple cells depict the ancilla regions. Black cells represent unused qubits.

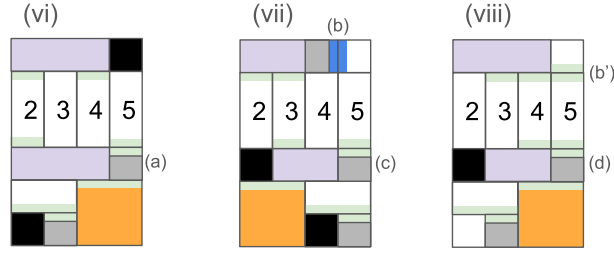


Figure 3.7: Multiple distillation trials. (a)–(d) represent distillation operations. (b') is a teleportation operation that transfers the magic state generated by (b) to qubit 5.

transferring the magic state out of the distillation factory. Once qubit 1 is reset, the magic state is teleported to the entangled qubit, making it available for the consumer's use.

The distillation process performs two rotations for each round, but it performs an additional teleportation operation in the first round, which may incur an extra time cost because the failure probability of zero-level distillation p_{fail} is too large, as discussed in Section 3.2.2. To mitigate the impact of these failures, we utilize unused qubits that are depicted as black-colored areas in Figure 3.6. Figure 3.7 illustrates the mitigation approach through distillation rounds 6, 7, and 8, where round 8 is the first round of the next distillation. For clarity, areas not related to qubit 5 are excluded from the figure. Given that qubit 5 is not involved in $\frac{\pi}{8}$ rotations performed during rounds 6 and 7, it is measured at the end of round 5, freeing up space for the subsequent distillation. The elements labeled (a), (b), (c), and (d) in the figure represent zero-level distillation instances, while (b') denotes a teleportation operation. Algorithm 1 details the execution of these operations. Because at least one of the distillation operations (a)–(d) must succeed to avoid a retry, the probability of having to retry becomes p_{fail}^4 , which is sufficiently small in our settings.

Algorithm 1 Execution of multiple distillation trials.

```

Perform (a) in round 6.
if (a) fails then
    Perform (b) in round 7.
    Perform (c) in round 7.
    if (c) succeeds then
        Do nothing.
    else if (b) succeeds then
        Perform (b') in round 8.
    else
        Perform (d) repeatedly until it succeeds in round 8.
    end if
end if

```

3.3 Performance evaluation

In this section, we conduct a comprehensive performance evaluation of (0+1)-level distillation and compare its performance with conventional protocols. We first assess the spacetime overhead of these protocols (Section 3.3.1), and then evaluate its impact on a critical application, namely Hamiltonian simulation (Section 3.3.2).

3.3.1 Spacetime overhead

To calculate the spacetime overhead associated with the distillation implementations, we employ Python and Mathematica scripts provided in Ref. [91], which are originally designed for analyses in Ref. [27]. For our specific evaluation of (0+1)-level distillation, we extend the Python script's functionality. This extension includes enhancing the calculation precision to accommodate high-fidelity settings, which are crucial for accurately assessing our distillation overhead.

Figure 3.8 illustrates the spacetime overhead associated with various distillation protocols at a physical error probability (p_{phys}) of 10^{-4} . The figure showcases four different distillation protocols, each represented by multiple points or stars, reflecting variations in specific parameters. For instance, the single-level 15-to-1 protocol has two parameters, d_X and d_Z ; configurations such as (15-to-1)_{7,3} and (15-to-1)_{9,3} are represented as individual blue points¹. The horizontal axis denotes the logical error probability of the distilled magic states, whereas the vertical axis quantifies the spacetime overhead, calculated as the product of the number of physical qubits and the cycle count. Notably, (0+1)-level distillation, which is shown as purple stars in the figure, significantly reduces the spacetime overhead for generating magic states within an error probability range of $[5 \times 10^{-17}, 10^{-11}]$. Specifically, it achieves a 70% reduction in overhead for magic states with an error probability around 10^{-13} , and a 65% reduction for those around 10^{-15} , compared to the (15-to-1)×(15-to-1) protocol.

The (15-to-1)×(20-to-4) protocol exhibits lower overhead than the (15-to-1)×(15-to-1) protocol in this range, yet (0+1)-level distillation presents even lower overhead overall. It is worth noting that

¹As noted in Section 2.4.1, the single-level 15-to-1 protocol actually has three parameters, d_X , d_Z , and d_m . We include parameter settings where $d_Z \neq d_m$ in the figure.

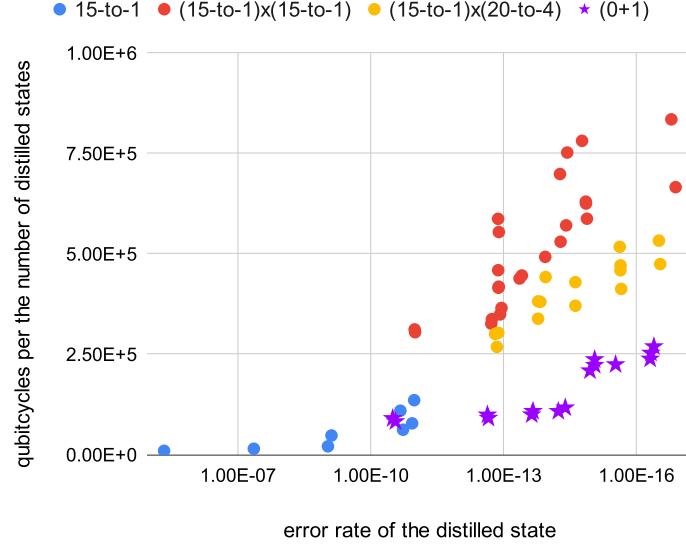


Figure 3.8: Spatiotemporal overhead of distillation protocols, $p_{\text{phys}} = 10^{-4}$.

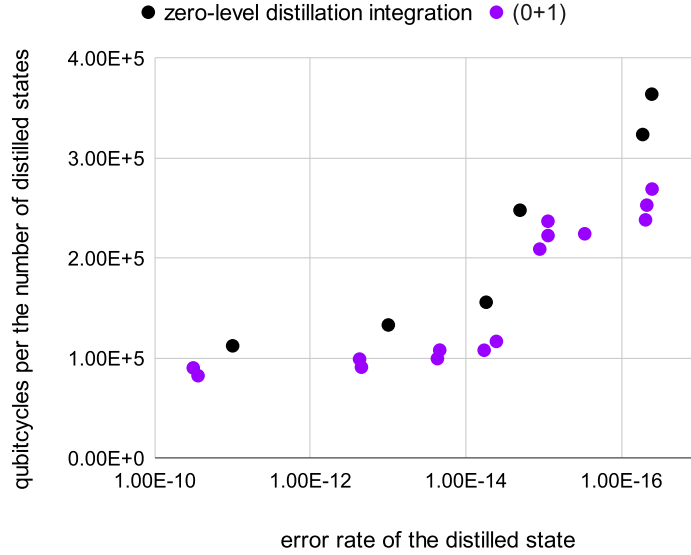


Figure 3.9: Performance gain of the second-level distillation refinements.

the $(15\text{-to-1}) \times (20\text{-to-4})$ protocol, which generates four magic states, faces challenges in the timely transfer of distilled magic states from the factory, often requiring additional spatial resources, which is a difficulty not encountered with $(0+1)$ -level distillation or the $(15\text{-to-1}) \times (15\text{-to-1})$ protocol.

In [Section 3.2](#), we described refinements to the second-level 15-to-1 implementation in $(0+1)$ -level distillation. [Figure 3.9](#) depicts the spacetime overhead of zero-level distillation integration (black points, [Section 3.2.1](#)) and $(0+1)$ -level distillation (purple points, [Section 3.2.4](#)), highlighting the effectiveness of the refinements.

One might ask whether the reduced overhead of $(0+1)$ -level distillation arises primarily from spatial or temporal reductions. In many cases, reductions in spatial overhead contribute more significantly than those in temporal overhead. For example, [Figure 3.10](#) compares the spatial

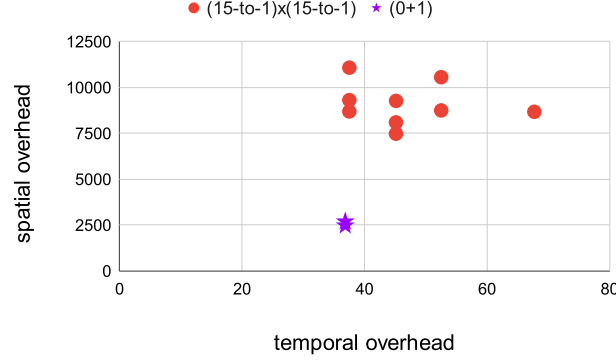


Figure 3.10: Spatial and temporal overhead of the $(15\text{-to-1}) \times (15\text{-to-1})$ protocol and $(0+1)$ -level distillation at error probabilities around 10^{-13} .

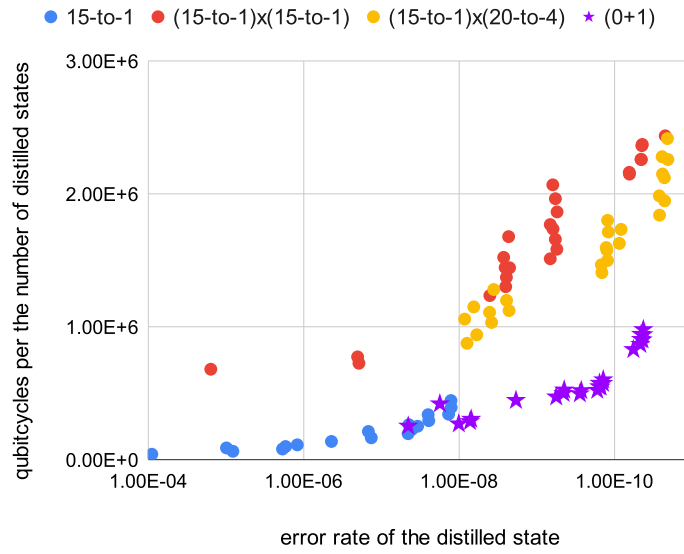


Figure 3.11: Spatiotemporal overhead of distillation protocols, $p_{\text{phys}} = 10^{-3}$.

overhead (in physical qubits) and the temporal overhead (in cycles) for both protocols at error probabilities around 10^{-13} . This comparison demonstrates that the spatial overhead of $(0+1)$ -level distillation is significantly lower than that of the $(15\text{-to-1}) \times (15\text{-to-1})$ protocol, while the differences in temporal overhead are less pronounced.

A similar trend is observed in Figure 3.11, which illustrates the spacetime overhead associated with the distillation protocols at a physical error probability (p_{phys}) of 10^{-3} . $(0+1)$ -level distillation presents favorable overhead for generating magic states within an error probability range of $[5 \times 10^{-11}, 10^{-8}]$, demonstrating its consistent efficiency.

3.3.2 Case study for massively parallel magic state consumption

In this subsection, we analyze the performance implications of the spacetime reduction observed in Section 3.3.1 on a quantum computation that consumes many magic states simultaneously. Specifically, we analyze the performance of a Hamiltonian simulation with Trotterization. This setup is similar to the Hamiltonian simulation described in Section 2.5.3, though we restate it here

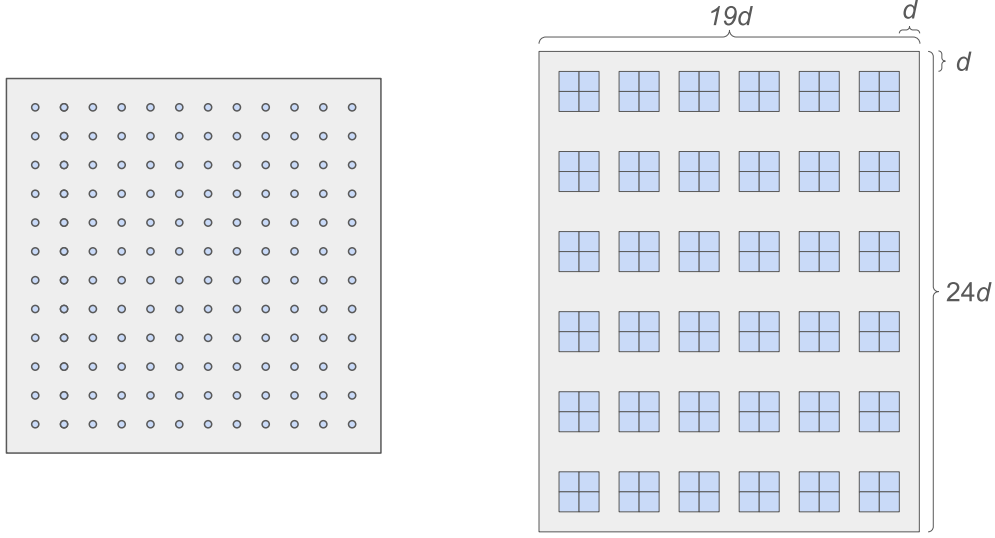


Figure 3.12: (left) Simulation target system. Each circle represents a site. (right) Qubit layout for the simulation. Each blue square is a logical data qubit corresponding to a site. Gray areas can be used to perform Pauli rotations.

since some details differ. Given a Hamiltonian $H = \sum_j P_j$, where each P_j is a Pauli operator, the time evolution of the target system can be approximated as

$$e^{-iHt} = (e^{\frac{-iHt}{M}})^M = (e^{\sum_j \frac{-it}{M} P_j})^M \approx (\prod_j e^{\frac{-it}{M} P_j})^M,$$

where M is a large integer. $\prod_j e^{\frac{-it}{M} P_j}$ is called a *Trotter step*, and M denotes the number of Trotter steps. Note that the order of the Pauli operators in a Trotter step does not matter:

$$e^{\frac{-it}{M} P_j} e^{\frac{-it}{M} P_k} \approx e^{\frac{-it}{M} (P_j + P_k)} = e^{\frac{-it}{M} (P_k + P_j)} \approx e^{\frac{-it}{M} P_k} e^{\frac{-it}{M} P_j}.$$

To conduct a thorough performance analysis of this computation, detailed information on the Hamiltonian $H = \sum_j P_j$ is required. In this subsection, let us assume that the target system consists of $12 \times 12 = 144$ square-shaped sites (Figure 3.12 (left)), and H is the sum of Pauli operators acting on every neighboring two sites. This means H is the sum of $12 \times 11 \times 2 = 264$ weight-two Pauli operators. Figure 3.12 (right) shows our arrangement of the logical qubits for the simulation. Let us call it the *central arena* because we will place magic state distillation factories around it. It consists of $24 \times 19 = 456$ $d \times d$ patches where d is the code distance.

Each $e^{\frac{-it}{M} P_j}$ is a Pauli rotation. With an efficient gate decomposition method [79], the number of $\pm \frac{\pi}{8}$ rotations required to implement $e^{\frac{-it}{M} P_j}$ is approximately $(-0.53 \log_2 \delta + 5)$ where δ is the precision of the rotation angle. Hence, the total number of magic states required for the simulation is approximately $(-140 \log_2 \delta + 1320)M$.

In theory, Pauli rotations acting on different qubits can be performed simultaneously, allowing for up to 72 rotations to be performed in parallel. In practice, this is not possible even with sufficient magic states, because the layout in Figure 3.12 (right) cannot supply the magic states to

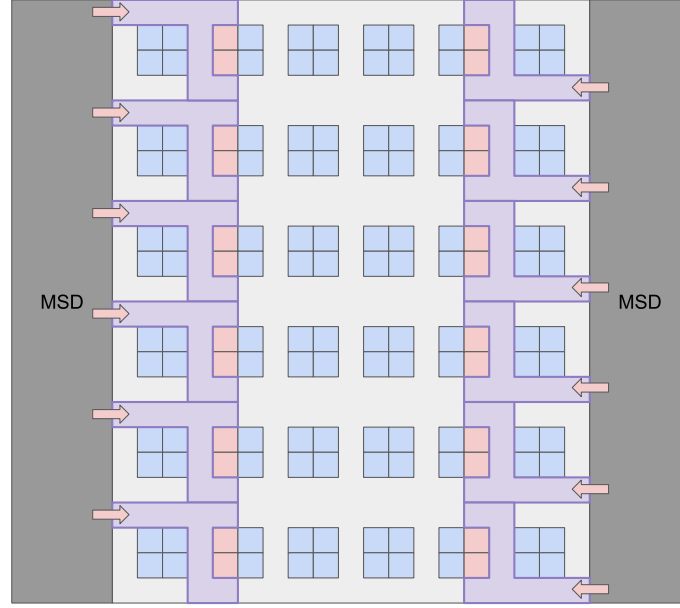


Figure 3.13: Qubit layout for the simulation with magic state distillation factories and twelve Pauli rotations performed simultaneously. Dark gray areas represent qubits used to distill magic states. Pink squares represent logical data qubits involved in Pauli rotations. Each purple area represents qubits used to perform the corresponding rotation. Each pink arrow represents a flow of a magic state used for the corresponding rotation.

all rotations simultaneously. As illustrated in Figure 3.13, it is possible to perform twelve rotations simultaneously, by providing six magic states from the left side and six magic states from the right side. With a magic state, a $\pm \frac{\pi}{8}$ rotation requires d cycles; hence, the total time cost of the computation is approximately $\frac{-140 \log_2 \delta + 1320}{12} M d$ cycles. Note that the time cost of Clifford gates in the decomposed Pauli rotations is negligible with the layout.

Now we are ready to set concrete parameters, determine the code distance, and choose appropriate parameters for magic state distillation protocols. Let us set $M = 10^6$, $\delta = 10^{-11}$, and $p_{\text{phys}} = 10^{-4}$. We also assume that we want the error probability of both Clifford gates and non-Clifford gates to be below 0.5%. Using the surface code error probability approximation [25, 26]

$$p_L(d) := 0.1(100p_{\text{phys}})^{\frac{d+1}{2}},$$

we have $456 \times \frac{-140 \log_2 \delta + 1320}{12} M d \times p_L(d) < 0.005$, and $d = 13$ is the smallest odd integer satisfying this inequality. The error probability of a magic state needs to be below $\frac{0.005}{-140M \log_2 \delta + 1320M} \approx 7.8 \times 10^{-13}$. Hence, we use the magic state distillation factories listed in Table 3.1. The rightmost column in the table represents the number of distillation factories required to provide sufficient magic states.

The spatial cost of the simulation, which is the total number of physical qubits required for the simulation, is the sum of the spatial cost of the central arena and the spatial cost of magic state distillation factories. The spatial cost of a magic state distillation factory is the spatial cost listed in Table 3.1. We also allocate one logical qubit for each $(15\text{-to-}1) \times (15\text{-to-}1)$ and $(0+1)$ -level

Table 3.1: Magic state distillation factories used for the simulation.

protocol	time	space	error probability	#MSD
$(15\text{-to-}1) \times (15\text{-to-}1)$	38	8690	1.9×10^{-13}	36
$(15\text{-to-}1) \times (20\text{-to-}4)$	70	15328	1.4×10^{-13}	18
(0+1)	37	2470	2.2×10^{-13}	36

distillation factory, and four logical qubits for each $(15\text{-to-}1) \times (20\text{-to-}4)$ factory, to store distilled magic states. In the distillation area in [Figure 3.13](#), some qubits are used to transfer magic states, but for simplicity of the analysis, we ignore this spatial cost. Magic state distillation is a probabilistic process, and parallel processing increases the impact of distillation failures. We can reduce the impact by placing additional distillation factories [\[33\]](#) or by placing a pool of magic states in each distillation factory [\[51\]](#). We also ignore this additional spatial cost for simplicity of the analysis.

The spatial cost of the simulation is 4.8×10^5 for the $(15\text{-to-}1) \times (15\text{-to-}1)$ protocol, 4.5×10^5 for the $(15\text{-to-}1) \times (20\text{-to-}4)$ protocol, and 2.6×10^5 for (0+1)-level distillation. Therefore, in this case, adopting (0+1)-level distillation reduces the total spatial cost by 47% compared to using the $(15\text{-to-}1) \times (15\text{-to-}1)$ protocol, and by 44% compared to using the $(15\text{-to-}1) \times (20\text{-to-}4)$ protocol. This significant reduction highlights the impact of efficiency of (0+1)-level distillation in computation that consumes many magic states simultaneously.

In [Table 3.1](#), (0+1)-level distillation offers 72% distillation spacetime overhead reduction compared to the $(15\text{-to-}1) \times (15\text{-to-}1)$ protocol, whereas it offers *only* 47% total spatial cost reduction. This is because the central arena ([Figure 3.12](#) (right)) is not affected by the protocol change. In other words, the spatial cost of magic state distillation factories relative to the total spatial cost limits the total spatial cost reduction.

[Figure 3.13](#) consumes twelve magic states simultaneously to reduce the time cost of the computation. We can further reduce the time cost by computing $e^{-i\theta Z} |+\rangle$ using magic states and gate decomposition in the dark gray areas in [Figure 3.13](#), and use state injection ([Figure 2.29](#)) to perform a Pauli rotation $e^{i\theta P}$. Although the number of required magic states is doubled on average, this configuration offers a much shorter execution time than the one discussed above, with sufficient magic states: the time cost is $\frac{264 \times 2}{12} Md$ cycles rather than $\frac{-140 \log_2 \delta + 1320}{12} Md$ cycles. Because this configuration consumes many more magic states in a shorter period of time, it requires many more distillation factories. As a result, the total spatial cost reduction offered by (0+1)-level distillation would be more significant.

We further estimate the performance under a larger physical error probability and a smaller number of Trotter steps. We set $M = 10^4$, $\delta = 10^{-9}$, and $p_{\text{phys}} = 10^{-3}$, which leads us to set $d = 23$. The error probability of a magic state needs to be below $\frac{0.005}{-140M \log_2 \delta + 1320M} \approx 9.1 \times 10^{-11}$, and we use magic state distillation factories listed in [Table 3.2](#).

The spatial cost of the simulation using the $(15\text{-to-}1) \times (15\text{-to-}1)$ protocol is 1.7×10^6 , whereas for the $(15\text{-to-}1) \times (20\text{-to-}4)$ protocol, it is 1.5×10^6 , and for (0+1)-level distillation, the cost is 9.7×10^5 . Therefore, in this setting, adopting (0+1)-level distillation reduces the total spatial cost by 41% compared to using the $(15\text{-to-}1) \times (15\text{-to-}1)$ protocol, and by 33% compared to the $(15\text{-to-}1) \times (20\text{-to-}4)$ protocol.

Table 3.2: Magic state distillation factories used for the simulation, $p_{\text{phys}} = 10^{-3}$.

protocol	time	space	error probability	# _{MSD}
(15-to-1)×(15-to-1)	92	23408	6.4×10^{-11}	48
(15-to-1)×(20-to-4)	130	50116	8.6×10^{-11}	18
(0+1)	78	10614	5.7×10^{-11}	42

3.4 Conclusion

In this chapter, we evaluated the spatial and temporal overhead of two-level distillation implementations, including those incorporating zero-level distillation, to understand how zero-level distillation contributes to reducing the overall distillation overhead. To this end, we introduced (0+1)-level distillation, a two-level distillation protocol that combines zero-level distillation and the 15-to-1 distillation protocol, and refines the second-level 15-to-1 distillation implementation to capitalize on the small footprint of zero-level distillation.

The numerical simulations presented in [Section 3.3.1](#) show that, under conditions of a physical error probability of $p_{\text{phys}} = 10^{-4}$ (10^{-3}) and targeting an error probability for the magic state within $[5 \times 10^{-17}, 10^{-11}]$ ($[5 \times 10^{-11}, 10^{-8}]$), (0+1)-level distillation reduces the spacetime overhead by more than 63% (61%) compared to the (15-to-1)×(15-to-1) protocol and more than 43% (44%) compared to the (15-to-1)×(20-to-4) protocol, demonstrating a substantial efficiency gain over the traditional protocols.

The resource estimation for a Hamiltonian simulation in [Section 3.3.2](#) shows that, with $p_{\text{phys}} = 10^{-4}$ (10^{-3}), adopting (0+1)-level distillation reduces the total spatial cost by 47% (41%) compared to using the (15-to-1)×(15-to-1) protocol, and by 44% (33%) compared to the (15-to-1)×(20-to-4) protocol. Furthermore, it is possible to consume more magic states simultaneously to reduce the time cost of the computation. Under such configurations, (0+1)-level distillation would offer an even more significant total spatial cost reduction.

Chapter 4

Efficient Magic State Cultivation with Lattice Surgery

Magic state distillation plays a crucial role in fault-tolerant quantum computation and represents a major bottleneck. In contrast to traditional logical-level distillation, physical-level distillation offers a significant overhead reduction by enabling direct implementation with physical gates. Magic state cultivation is a state-of-the-art physical-level distillation protocol that is compatible with the square-grid connectivity and yields high-fidelity magic states. However, it relies on the complex grafted code, which incurs substantial spacetime overhead and complicates practical implementation. In this chapter, we propose an efficient cultivation-based protocol compatible with the square-grid connectivity. We reduce the spatial overhead by avoiding the grafted code and further reduce the average spacetime overhead by utilizing code expansion and enabling early rejection. Numerical simulations show that, with a color code distance of 3 and a physical error probability of 10^{-3} , our protocol achieves a logical error probability for the resulting magic state comparable to that of magic state cultivation ($\approx 3 \times 10^{-6}$), while requiring about half the spacetime overhead. Our scheme provides an efficient and simple distillation protocol suitable for megascale use cases and early fault-tolerant devices. This chapter is based on “**Efficient magic state cultivation with lattice surgery**,” Hirano, Toshio, Itogawa, Fujii, *arXiv: 2510.24615* [50].

4.1 Introduction

Quantum computers promise to solve problems of practical importance that are intractable for classical computers, such as simulating quantum systems [4] and integer factoring [3]. Existing quantum computers are highly susceptible to noise, which prevents them from executing complex algorithms such as Shor’s factoring algorithm. To execute such complex quantum algorithms, we need fault-tolerant quantum computers (FTQCs) [9], which rely on quantum error correction (QEC) [10, 57].

In FTQC, each *logical gate* must be implemented in a fault-tolerant manner. While fault-tolerant implementations of Clifford gates are relatively inexpensive, Clifford gates alone are not sufficient for universal quantum computation [13]. Implementing a non-Clifford gate such as the T gate

requires a special quantum state, a *magic state*, which is obtained through a process known as *magic state distillation* [14] (Section 2.4). Magic state distillation is resource-intensive and remains a major bottleneck in fault-tolerant quantum computation. Consequently, many distillation protocols have been proposed [14, 20–29, 38, 49, 92–94].

Bravyi *et al.* [14] proposed the 15-to-1 distillation protocol, which exploits T ’s transversality on the $[[15, 1, 3]]$ code. Earlier, predating the term “magic state distillation”, Knill *et al.* [20] proposed a protocol based on the Hadamard test of the H gate, relying on H ’s transversality on the $[[7, 1, 3]]$ code. These studies formulated magic state distillation as a quantum algorithm built from ideal Clifford gates with noisy inputs, and this formulation was followed by subsequent studies [21, 22, 24–26]. Running such algorithms on FTQC requires protecting each gate against noise using QEC, incurring a significant cost.

Goto [23] proposed a technique called *physical-level distillation*. In contrast to the above formulation, the distillation circuit operates with noisy Clifford gates. Specifically, the circuit detects any single gate error, thereby achieving a logical error probability of $\mathcal{O}(p_{\text{phys}}^2)$, where p_{phys} is the physical error probability. To this end, the distillation protocol consists of a Knill-style Hadamard test with a flag qubit. Its physical-gate implementation significantly reduces the distillation overhead.

Goto’s protocol requires all-to-all connectivity. Chamberland *et al.* [28] and Itogawa *et al.* [29] proposed physical-level distillation protocols that work with planar connectivity. The latter, known as zero-level distillation (Section 2.4.3), operates on the simpler square-grid connectivity and yields a magic state encoded in a surface code [30, 31] (Section 2.2.4), achieving a logical error probability of 10^{-4} for a realistic $p_{\text{phys}} = 10^{-3}$ setting. Gidney *et al.* [38] refined these works and proposed *magic state cultivation* (MSC) (Section 2.4.4), which works with the square-grid connectivity. This protocol extensively utilizes postselection to produce high-fidelity magic states. It yields a magic state with a logical error probability of 2×10^{-9} at $p_{\text{phys}} = 10^{-3}$. MSC prepares a magic state encoded in a 2D color code [39], which is then transformed into the grafted code, another QEC code compatible with matching QEC decoders. However, due to its high-weight stabilizer checks, the grafted code is less space-efficient than the rotated surface code [40] (Section 2.2.4) with a comparable logical error probability, incurring substantial spacetime overhead.

Although these physical-level distillation protocols significantly reduce the distillation overhead, further advances are desirable, particularly when many magic states are consumed simultaneously during the parallel execution of multiple T gates [33, 52]. While this has motivated the development of various MSC-based protocols [92–94], they relax the connectivity restriction that MSC respects. This restriction is particularly important for superconducting devices. Although platforms such as neutral atoms allow flexible connectivity, qubit shuttling incurs temporal and idling costs; thus, maintaining local operations on a fixed lattice can offer temporal advantages. Therefore, there is a need for a more efficient magic state distillation protocol compatible with the square-grid connectivity.

In this chapter, we propose a novel MSC-based protocol that significantly reduces the spacetime overhead of distillation without relaxing the hardware connectivity requirements. We avoid using the grafted code by employing lattice surgery [41, 42] (Section 2.2.5), as in the original zero-level distillation [29], to transfer the magic state from the color code to the rotated surface code. Lattice

surgery and the rotated surface code allow us to employ two techniques to further reduce the distillation overhead: (1) utilizing surface code expansion [95] to minimize the qubit count in the early stages of distillation, and (2) predicting the logical error probability using a simple lookup table, allowing early rejection to reduce the average spacetime overhead.

In addition to efficiency, our protocol provides simplicity. The grafted code is complex, and avoiding its implementation is appealing to practitioners. The rotated surface code and lattice surgery are simpler, already supported by existing compilers and instruction sets, and impose little additional burden on control hardware. Thus, our protocol can leverage existing software and hardware infrastructures, making it more practical for near-term implementations.

We conducted numerical simulations using Stim [96] to evaluate the performance of our protocol. With a color code distance of 3, our protocol achieves a logical error probability for the resulting magic state comparable to that of MSC, while requiring about half the spacetime overhead. Specifically, at $p_{\text{phys}} = 10^{-3}$, our protocol achieves a logical error probability of 1.6×10^{-6} for $S|+\rangle$ distillation. This matches the 1.2×10^{-6} achieved by MSC with color code distance 3, and corresponds to a logical error probability of 3×10^{-6} for $T|+\rangle$ distillation. For logical error probabilities in the range $[10^{-6}, 10^{-5}]$, our protocol requires less than half the spacetime overhead needed for MSC. These results demonstrate that our approach offers a practical path toward more efficient magic state distillation in fault-tolerant quantum computation, especially for megaquop use cases [48] (computation involving $\sim 10^6$ logical gate operations) and early fault-tolerant devices [34–36] (systems with 10^3 – 10^5 physical qubits).

The rest of this chapter is structured as follows. In Section 4.2, we detail our proposal: magic state cultivation with lattice surgery. In Section 4.3, we present a performance evaluation of the protocol, including the numerical simulation settings and results. Finally, Section 4.4 summarizes our findings.

4.2 Magic state cultivation with lattice surgery

In this section, we describe our proposed protocol: *magic state cultivation with lattice surgery* (MSC-LS). Figure 4.1 depicts the protocol with $d_{\text{color}} = 3$. The protocol shares the injection and cultivation stages with the original magic state cultivation (MSC) [38] (Section 2.4.4). Afterward, the magic state is transferred to the rotated surface code [40] (Section 2.2.4) of distance $d_{\text{intermediate}}$ using lattice surgery [41, 42] (Section 2.2.5). This transfer requires d_{color} rounds of syndrome extraction. At the end of the lattice surgery procedure, we perform additional postselection using a lookup table that stores error syndromes that are available at the stage. We then perform code expansion, followed by d_{color} rounds of syndrome extraction to stabilize the code. Finally, we compute the complementary gap [55] (Section 2.2.6) and perform postselection based on its value. We perform r rounds of syndrome extraction while waiting for the QEC decoder to compute the complementary gap. Similar to the magic state cultivation case, the fault distance of the resulting magic state depends on the complementary gap threshold.

Our protocol differs from MSC in three key aspects: (1) adopting the space-efficient rotated surface code as the final QEC code, (2) using an intermediate code distance during the lattice surgery

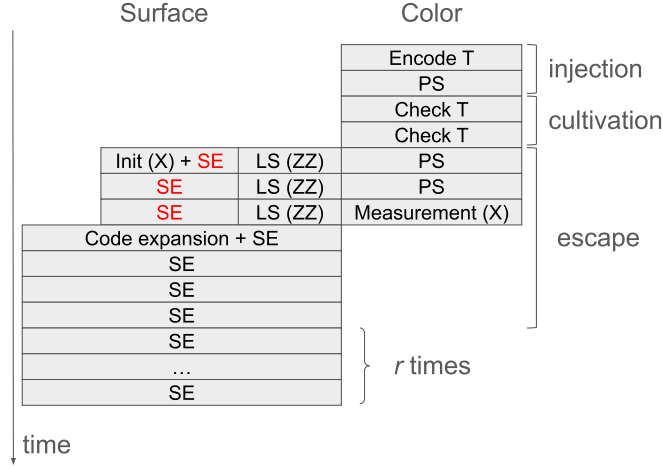


Figure 4.1: Magic state cultivation with lattice surgery (MSC-LS) for $d_{\text{color}} = 3$. The left area represents the spatial resource (physical qubits) used by the rotated surface code, and the right area represents those used by the color code. “Encode T” denotes the non-fault-tolerant encoding of a magic state into the distance-3 2D color code. Two consecutive “Check T” denote the application of two Hadamard tests. “LS (ZZ)” denotes the ZZ measurement implemented by lattice surgery. “PS” denotes postselection, and “SE” denotes syndrome extraction. The detectors obtained from the syndrome extractions marked in red are used for early rejection via the lookup table.

procedure to reduce the spatial overhead in the early stages, and (3) incorporating the lookup table to enable early rejection. Although additional d_{color} rounds of syndrome extraction are needed for code stabilization after code expansion, the spatial overhead reduction and early rejection outweigh the additional time overhead, as shown later in Section 4.3.

We detail the lattice surgery and code expansion procedures in the following subsections. We describe how to construct and use the lookup table in Section 4.2.3.

4.2.1 Lattice surgery

In the lattice surgery procedure, we teleport the magic state from the 2D color code to the rotated surface code of distance $d_{\text{intermediate}}$, following the spirit of the zero-level distillation protocol [29]. Specifically, we encode $|+\rangle_L$ into the rotated surface code, perform a $Z_L Z_L$ measurement via lattice surgery, and finally measure all physical qubits of the color code in the X basis.

In this subsection, we focus on the case where $d_{\text{color}} = 3$, and we leave the implementations for larger color distances to future work. We denote the physical qubits on the color code as qubits 0–6, and physical qubits on the top edge of the rotated surface code as a – e (Figure 4.2). Figure 4.3 illustrates the sequence for teleporting a magic state from the color code of distance 3 to the rotated surface code of distance 5. In the first step, the magic state is encoded in the color code prepared during the injection and cultivation stages discussed in Section 2.4.4, and each physical qubit in the rotated surface code is initialized to $|+\rangle$ (Figure 4.3 (a)).

In the second step, Z stabilizer checks on the merged boundaries are measured (Figure 4.3 (b)). Each measurement is performed three times to detect measurement errors. Meanwhile, stabilizer checks on both codes are measured. Note that, during the construction of the merged

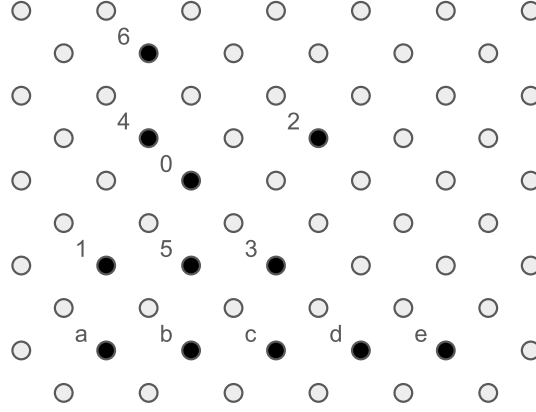


Figure 4.2: Physical qubit positions. Qubits 0–6 are data qubits of the distance-3 2D color code, and qubits a – e are data qubits on the top edge of the rotated surface code.

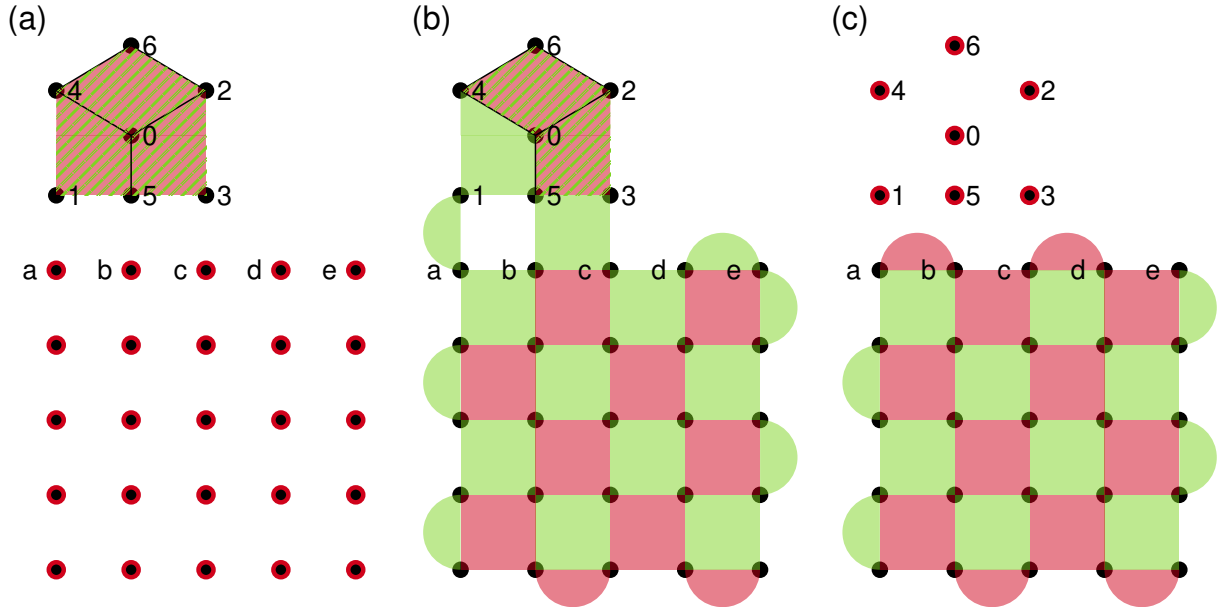


Figure 4.3: Lattice surgery operation to teleport a magic state from the 2D color code of distance 3 to the rotated surface code of distance 5. X stabilizer checks are colored red, and Z stabilizer checks are colored green. A red circle on a physical qubit represents initialization or measurement of a physical qubit in the X basis. (a) The magic state is encoded in the color code, and each physical qubit in the rotated surface code is initialized to $|+\rangle$. (b) Z stabilizer checks on the merged boundaries (Z_{1a} , Z_{35bc} , and Z_{de}) are measured three times. Meanwhile, all stabilizer checks on both codes are measured, except X_{0145ab} . (c) Physical qubits on the color code are measured in the X basis. The X stabilizer checks on the top boundary of the surface code (X_{ab} and X_{cd}) are recovered. X stabilizer checks X_{0145ab} , X_{0235} , and X_{0246} are measured, and the outcomes are subject to postselection. X_{146} is the logical X measurement outcome.

code, a four-weight X stabilizer operator X_{0145} becomes a six-weight X stabilizer operator X_{0145ab} because X_{0145} and Z_{1a} do not commute. Since performing a six-weight syndrome extraction is costly, we skip syndrome extraction for X_{0145ab} .

Lastly, physical qubits on the color code are measured in the X basis (Figure 4.3 (c)). We also recover the X stabilizer checks along the top edge of the surface code. These yield X syndrome

measurement outcomes X_{0145ab} , X_{0235} , and X_{0246} , along with the logical X measurement outcome $X_L = X_{146}$. If the logical X measurement outcome is 1, then we apply the logical Z operator to the magic state encoded in the rotated surface code. This can be performed classically using the Pauli feedforward mechanism [59].

In the second step, we use the superdense cycle [97] for syndrome extraction on the color code to measure corresponding X and Z stabilizer checks simultaneously. Specifically, in the first syndrome extraction round, we check the following stabilizer checks: (1) Z_{0145} , (2) Z_{0235} and X_{0235} , and (3) Z_{0246} and X_{0246} . In the second round, we check (4) Z_{0145} , (5) Z_{0235} , and (6) Z_{0246} . In the third step of the lattice surgery procedure, we derive (7) X_{0145ab} , (8) X_{0235} , and (9) X_{0246} from destructive X measurement outcomes on the color code and the syndrome measurement outcome of X_{ab} . In summary, we perform two rounds of syndrome extraction for each stabilizer check on the color code except X_{0145} , and one round for X_{0145ab} in this procedure.

During the sequence, color code syndrome extractions are performed in an error-detecting manner to preserve the fault distance of the magic state. This means that if any error syndrome is nontrivial, the distillation process is rejected. The same applies to the parities of the Z stabilizer checks on the merged boundaries. By contrast, error correction is performed for stabilizer checks on the rotated surface code. This is justified when $d_{\text{intermediate}}$ is larger than the fault distance. To make the detector error model compatible with matching decoders (Section 2.2.2), we modify the noise model on the color code qubits, assigning zero error probability to errors on these physical qubits.

The initial measurement outcomes for the X stabilizer checks introduced in the last step of lattice surgery (except for X_{ab}) are non-deterministic. Hence, extra rounds of syndrome extraction are required to stabilize the code; in Figure 4.1, three rounds of syndrome extraction are performed after code expansion. This is different from the grafted code case, where all the syndrome measurement outcomes are deterministic at the end of the escape stage, and thus no additional rounds are needed (Figure 2.31).

4.2.2 Code expansion

In the code expansion procedure, we increase the distance of the rotated surface code from $d_{\text{intermediate}}$ to d_{final} . We use Li's protocol [95] for the rotated surface code. As illustrated in Figure 4.4, after performing lattice surgery, the surface code patch can be expanded upward or downward. Upward expansion is more efficient in terms of packing, because we can reuse the physical qubits used for the color code for the final surface code patch (with a sufficiently large d_{final}), though its implementation is more involved. For simplicity of implementation, in this thesis, we focus on the downward expansion and leave upward expansion for future work.

Figure 4.5 illustrates the code expansion sequence. We initialize each physical qubit to $|0\rangle$ or $|+\rangle$, depending on its location, and perform surface code syndrome extraction.

Similar to the discussion in Section 4.2.1, because the initial syndrome measurement outcomes of some of the newly introduced stabilizer checks are non-deterministic, several rounds of surface code syndrome extraction are needed to stabilize the code. This can also be done in parallel with

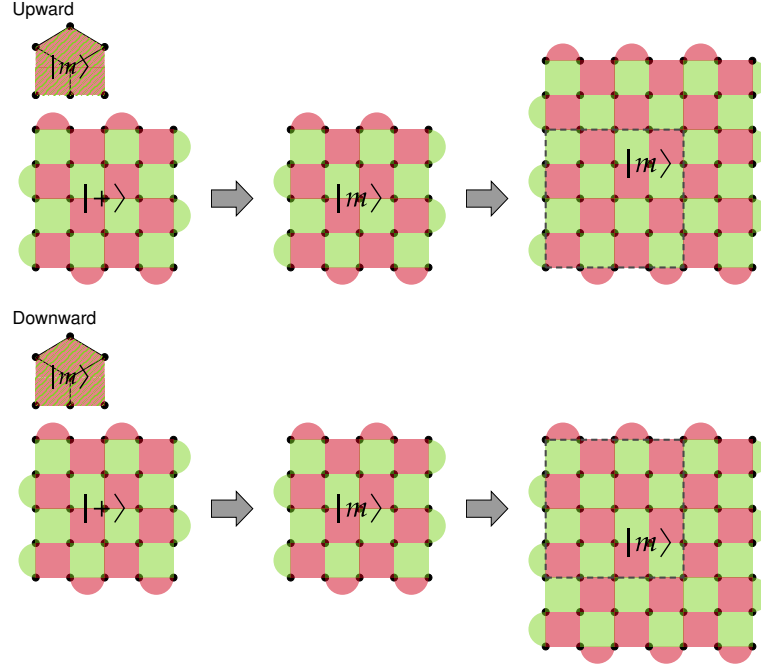


Figure 4.4: Upward surface code expansion (top) and downward surface code expansion (bottom). $|m\rangle$ denotes a magic state.

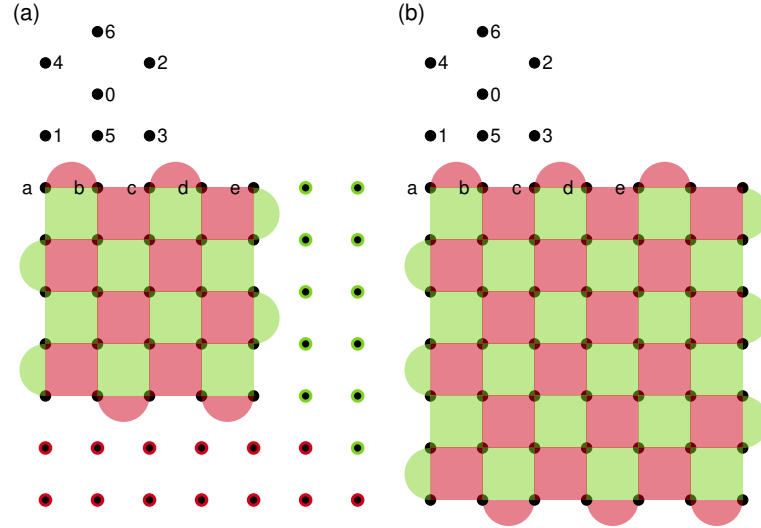


Figure 4.5: Surface code expansion from distance 5 to distance 7. A red (green) circle on a physical qubit represents a $|+\rangle$ ($|0\rangle$) initialization. (a) Additional physical qubits are initialized to $|0\rangle$ or $|+\rangle$ according to their locations. (b) Additional stabilizer checks are placed.

the code stabilization required for lattice surgery. Hence, we perform d_{color} rounds of syndrome extraction after code expansion (see Figure 4.1 for an example).

4.2.3 Lookup table

After code expansion, we perform d_{color} rounds of syndrome extraction to stabilize the code, and r rounds of syndrome extraction while waiting for the decoder to compute the complementary gap.

Algorithm 2 Construct a lookup table.

```

table = {}
k ← the size of a syndrome obtained before code expansion
for shot in shots do
    synd ← the error syndrome for shot
    pattern ← synd[0..k]
    gap ← the complementary gap of synd
    if table has pattern then
        (num_valid, num_invalid) ← table.get(pattern)
    else
        (num_valid, num_invalid) ← (0, 0)
    end if
    valid ← true if the resulting magic state is correct
    if gap ≥  $g_{\text{th}}$  and valid then
        num_valid += 1
    else
        num_invalid += 1
    end if
    table.set(pattern, (num_valid, num_invalid))
end for
Remove entries from table where num_valid > 0.
Remove entries from table where num_invalid <  $c_{\text{th}}$ .
return table

```

In this subsection, we describe the lookup table, a data structure used to reject distillation attempts before code expansion. This reduces the average distillation overhead by allowing us to reject such attempts without waiting for $d_{\text{color}} + r$ rounds.

A lookup table contains syndromes obtained prior to code expansion that would likely be rejected by the final postselection. At the end of the lattice surgery procedure (before code stabilization), we check whether the obtained error syndrome appears in the table; if so, we reject the distillation attempt. This is equivalent to predicting whether the logical error probability of the resulting magic state is larger than a certain threshold given the error syndrome.

We expect that whether an error syndrome appears in a lookup table can be checked immediately, because it is significantly simpler than calculating the complementary gap. Of course, the feasibility of the lookup table depends on the number of entries and the accuracy of the predictions. We note that our protocol is suitable for the lookup table approach, since $d_{\text{intermediate}}$ is generally small and the error syndromes obtained prior to code expansion are limited in length, resulting in fewer table entries.

Algorithm 2 presents the algorithm for constructing a lookup table. The algorithm has three inputs: a list of distillation attempts *shots*, the complementary gap threshold g_{th} , and the threshold on the number of invalid cases c_{th} .

When the target logical error probability is very small (equivalently, when the target complementary gap is large), it may be that any distillation attempt with a nontrivial syndrome before the code expansion should be rejected. In this case, we do not store entries in the lookup table. We

instead set a boolean flag `remove_all_nontrivial` in the lookup table. If the flag is set, the lookup table rejects all nontrivial syndromes.

The lookup table allows us to predict whether the logical error probability of the resulting magic state exceeds a certain threshold, given an error syndrome, while requiring far less computation than calculating the complementary gap. More sophisticated logic than a simple existence check may further reduce memory requirements and improve prediction accuracy. In this thesis, we employ this simple implementation and leave further investigation for future work. As shown later in [Section 4.3](#), under our simulation setting with a high complementary gap threshold, a lookup table with 160 thousand entries reduces the average spacetime overhead of distillation by 15%. For more memory-constrained systems, a smaller lookup table can be used.

4.3 Numerical simulation

This section presents a performance evaluation of magic state cultivation with lattice surgery (MSC-LS). We use Stim [96] and PyMatching [98] for the evaluation. Our implementation is available at <https://github.com/yutakahirano/msc-ls/>. Because Stim can only simulate Clifford circuits, we replace T gates in the injection and cultivation stages with S gates. This modification causes the distillation circuit to output $S|+\rangle$ instead of $T|+\rangle$. Gidney *et al.* [38] conjecture that cultivating $T|+\rangle$ has a logical error probability approximately twice as high as cultivating $S|+\rangle$, which we rely on.

4.3.1 Simulation settings

We evaluate the performance of MSC-LS using several parameters. These include the color code distance $d_{\text{color}} = 3$, the intermediate rotated surface code distance $d_{\text{intermediate}}$, the final rotated surface code distance d_{final} , and the number of rounds of syndrome extraction r performed while waiting for the complementary gap computation. Simulations are carried out under circuit-level uniform depolarizing noise of strength p_{phys} . Unless otherwise specified, we set $d_{\text{final}} = 11$, $r = 10$, and $p_{\text{phys}} = 10^{-3}$. For the original magic state cultivation, we use a grafted code distance of $d_{\text{grafted}} = 15$, following the observation of Gidney *et al.* [38] that the logical error probability of the distance-15 grafted code is comparable to that of the distance-11 rotated surface code.

We take a “round” as the unit of distillation time. A round is a series of circuit layers that corresponds to one round of syndrome extraction in the 2D color code or the rotated surface code. This unit allows us to compare the spacetime overhead of distillation with magic state cultivation. [Algorithm 3](#) presents the algorithm for calculating the spacetime overhead of distillation.

4.3.2 Results

[Figure 4.6](#) illustrates X and Z logical error probabilities resulting from the lattice surgery procedure described in [Section 4.2.1](#) as a function of p_{phys} . Here, to focus on the logical errors caused by lattice surgery, the simulation runs the following sequence. First, an ideal $|+\rangle_L$ ($|0\rangle_L$) state is encoded in the distance-3 2D color code. Next, the lattice surgery procedure described in [Section 4.2.1](#) is

Algorithm 3 Calculate the spacetime overhead.

```

 $cost \leftarrow 0$ 
for each round do
   $q \leftarrow$  the number of qubits used in this round
   $p \leftarrow$  the success probability of this round
   $cost \leftarrow (cost + q)/p$ 
end for
return  $cost$ 

```

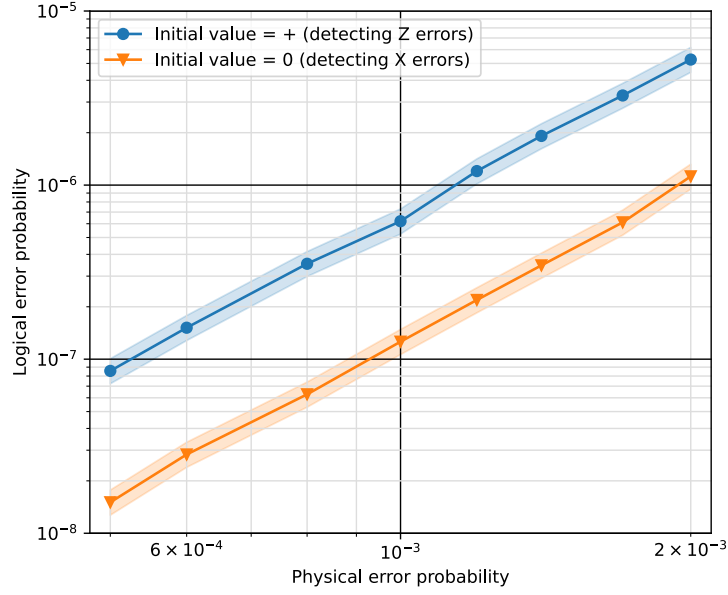


Figure 4.6: Logical X and Z error probabilities introduced by the lattice surgery procedure.

performed to teleport the quantum state to the rotated surface code of distance $d_{\text{intermediate}} = 5$. Finally, we apply one round of noise-free syndrome extraction on the rotated surface code, followed by postselection and a noise-free logical measurement in the X (Z) basis, after which we verify the outcome. This evaluates the Z (X) logical error probability of the procedure. Fitting the data to the form $C p_{\text{phys}}^\alpha$ gives $\alpha \approx 2.92$ for Z errors and $\alpha \approx 3.30$ for X errors, indicating that the procedure preserves a fault distance of 3, consistent with expectations.

Figure 4.7 illustrates the trade-off between the logical error probability and end-to-end distillation success probability of MSC-LS and MSC. Each point in a series corresponds to a different complementary gap threshold used in the final postselection. MSC-LS matches MSC in logical error probability but outperforms it in end-to-end distillation success probability. Moreover, MSC-LS with $d_{\text{intermediate}} = 5$ outperforms MSC-LS with $d_{\text{intermediate}} = 3$.

We construct lookup tables using Algorithm 2 with various complementary gap thresholds g_{th} . For each g_{th} setting, we sample 10^{10} shots, with the threshold for invalid cases set to $c_{\text{th}} = 100$. Figure 4.8 shows the coverage of the lookup table with $g_{\text{th}} = 15.48$. At maximum size, the table contains 159,321 entries and rejects 9.6% of distillation attempts. Figure 4.9 and Figure 4.10 show the qubit usage and survival probability during distillation. Without the lookup table (Figure 4.9),

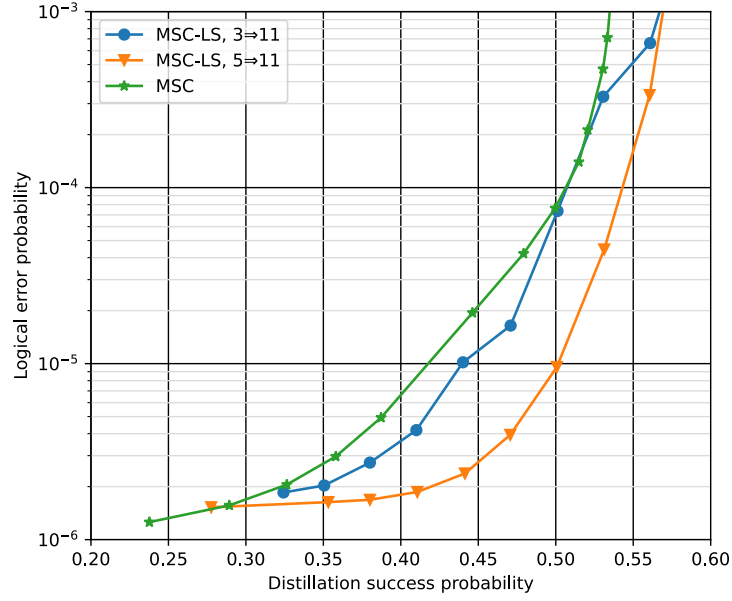


Figure 4.7: Logical error probability of the resulting magic state as a function of distillation success probability. Blue: MSC-LS with $d_{\text{intermediate}} = 3$; orange: MSC-LS with $d_{\text{intermediate}} = 5$; green: MSC. Here we do not use the lookup table for MSC-LS.

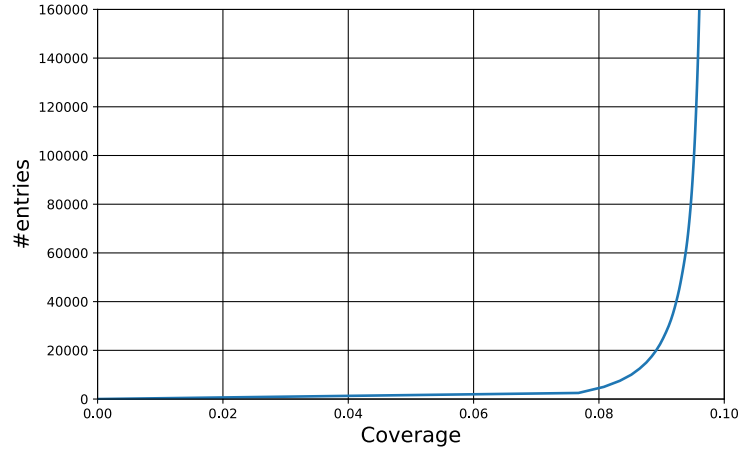


Figure 4.8: Coverage of the lookup table. The horizontal axis denotes the ratio of distillation attempts that would be additionally rejected by the lookup table. The vertical axis represents the number of entries in the table.

43% of samples are rejected before code expansion, 22% by the final postselection, and 35% are accepted. By contrast, with the lookup table (Figure 4.10), 52% of samples are rejected before code expansion, 13% are rejected by the final postselection, and 35% are accepted. This early rejection reduces the spacetime overhead by 15% in this setting.

Figure 4.11 depicts the spacetime overhead of distillation of MSC-LS and MSC. It shows that in the logical error probability range of $[10^{-6}, 10^{-5}]$, MSC-LS requires less than half the spacetime overhead required for MSC. For example, MSC-LS reduces the spacetime overhead by 59% (55%)

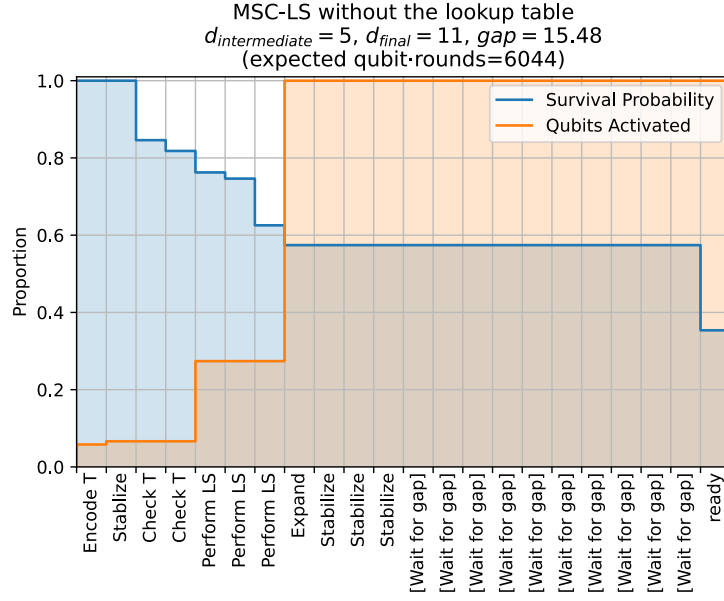


Figure 4.9: Qubit usage and survival probability during distillation without the lookup table. Each column represents a round. The blue region represents the survival probability of distillation. The orange region represents the physical qubit usage normalized to the maximum qubit usage (241 qubits in this case).

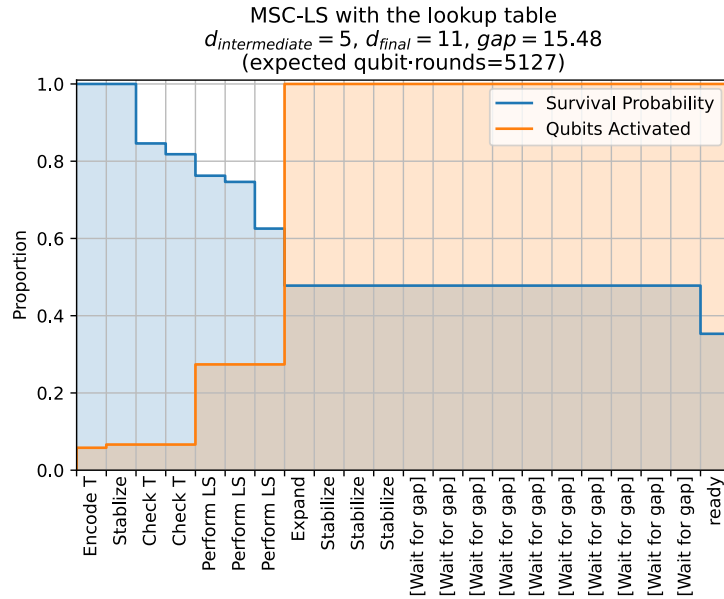


Figure 4.10: Qubit usage and survival probability during distillation with the lookup table, using the same color scheme as Figure 4.9.

when distilling an $S|+\rangle$ state with a logical error probability of 2×10^{-6} (4×10^{-6}). The lookup table performs more effectively in the lower range of logical error probability, which is expected, given that postselection is performed more strongly in this range.

Figure 4.8 also shows that the lookup table is useful even when the number of entries in the

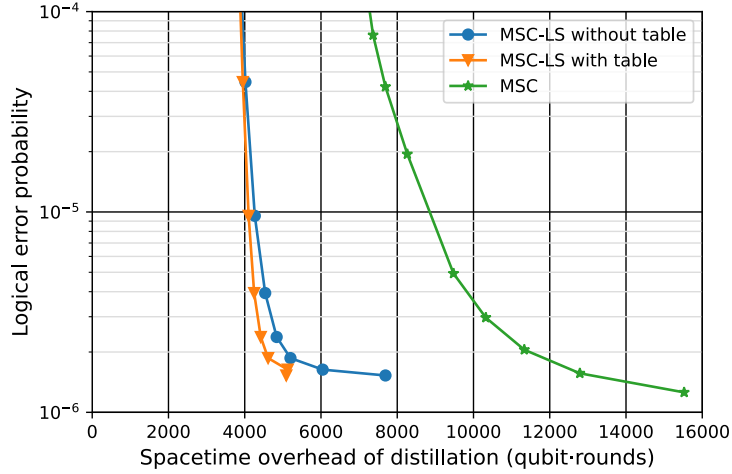


Figure 4.11: Spacetime overhead of MSC-LS with and without the lookup table and MSC.

table is limited. For example, with only 5,000 entries, the coverage reaches 8%. Hence, the lookup table is useful even in memory-constrained systems. The number of detectors contained in a table entry is 54 with our simulation setting, and hence each table entry requires only 54 bits and is very lightweight.

4.4 Conclusion

In this chapter, we proposed magic state cultivation with lattice surgery, an efficient magic state distillation protocol. The efficiency of our protocol arises from (1) adopting the space-efficient rotated surface code as the final QEC code, (2) using an intermediate code distance smaller than d_{final} during the lattice surgery procedure to reduce the spatial overhead in early stages, and (3) incorporating the lookup table to enable early rejection. Our protocol also provides simplicity, as it avoids the complex grafting procedure. Numerical simulations show that our protocol achieves a logical error probability for the resulting magic state comparable to that of magic state cultivation, while requiring only about half the spacetime overhead of magic state cultivation. Although we leave implementations with larger color code distances for future work, distillation with color code distance 3 remains useful for early applications of FTQC devices [34–36], for use in further distillation [27, 49], or under lower physical error probabilities. Therefore, we conclude that magic state cultivation with lattice surgery offers a practical path to more efficient magic state distillation in fault-tolerant quantum computation, thereby reducing the total spacetime cost of T -intensive quantum computation.

In this chapter, we focused on implementations with $d_{\text{color}} = 3$. We see no fundamental obstacles to implementations at larger color distances. However, whether such implementations are more efficient than magic state cultivation with the corresponding color distance remains open and is an important direction for future work.

Another important direction for future work is to develop a more concrete distillation pipeline. Both magic state cultivations with and without lattice surgery have relatively low end-to-end success

probabilities, and the average spacetime cost of distillation is evaluated under the assumption that distillation can be parallelized and pipelined. For instance, satisfying this assumption requires additional code expansion schemes beyond the downward expansion discussed in [Section 4.2.2](#). This capability is critical for using the protocol in compilation schemes that require a small, on-demand distillation factory [\[52\]](#).

Chapter 5

MagicPool: Dealing with Magic State Distillation Failures on Large-Scale Fault-Tolerant Quantum Computer

Magic state distillation, which is a probabilistic process used to generate magic states, plays an important role in universal fault-tolerant quantum computers. To solve practically important problems, we need to run complex programs on fault-tolerant quantum computers, and we need to use hardware resources efficiently. Exploiting parallelism is a major optimization strategy, and compilers are responsible for performing optimizations to allow parallel processing. However, the probabilistic nature of magic state distillation is not compatible with compile-time optimizations and results in an additional runtime delay. In this chapter, we propose a scheme to mitigate the negative performance impact caused by magic state distillation failures: specifically, we maintain a pool of magic states. We ran simulations of quantum circuits to verify the magnitude of the runtime delay and the effectiveness of the mitigation approach. The simulation results show that the runtime delay is amplified by parallel processing, and pooling effectively reduces the runtime delay with a small spatial cost. This chapter is based on “**MagicPool: Dealing with Magic State Distillation Failures on Large-Scale Fault-Tolerant Quantum Computer,**” Hirano, Suzuki, Fujii, *arXiv: 2407.07394* [51].

5.1 Introduction

Quantum computers promise to solve practically important problems that are believed to be difficult for classical computers, such as quantum physics simulation [4] and integer factoring [3]. Currently available quantum computers are too susceptible to noise to run quantum algorithms for these applications; hence, to fully realize this promise, we need Fault-Tolerant Quantum Computers (FTQCs) [9] built on top of error-correcting codes [10, 57]. Surface codes [30, 69] (Section 2.2.4) are among the most promising quantum error-correcting codes because they offer a relatively high error threshold using only nearest-neighbor connectivity. Experimental quantum error correction has been demonstrated based on the surface-code architecture using quantum devices with nearest-neighbor

interactions, such as superconducting qubits [11]. The next milestone is to perform fault-tolerant quantum computation with error-corrected logical qubits.

A major bottleneck in FTQC with surface codes is the application of non-Clifford gates such as T and CCZ , since they require special quantum states called *magic states*, which are generated with computationally expensive, probabilistic protocols known as *magic state distillation* [14] (Section 2.4). In contrast, other logical operations, such as Clifford gates and Pauli measurements, are relatively inexpensive. Thus, efficient handling of non-Clifford gates has been explored as one of the most important issues. These efforts can be categorized into two approaches: the production side and the consumption side. On the production side, methods to reduce the spatial and temporal overhead of magic state distillation have been extensively studied [24–27, 29, 38, 90, 99]. One of the most successful optimization directions has been to use postselection to reduce the required resources at the cost of small success probabilities. This technique is particularly important in the early era of FTQCs, where the available qubit count is strictly limited. On the consumption side, many approaches have been proposed that aim to maximize the parallelism of operations that consume magic states, which is equivalent to minimizing T depth [84, 100–102]. The performance of an FTQC is maximized when the production and consumption optimizations are well balanced.

However, there is an overlooked point that must be considered in interfacing efforts on these two sides: scheduling probabilistic protocols under high parallelism. Once an abstract quantum circuit with low T depth is given, FTQC compilers translate it into a sequence of operations. In this translation, FTQC compilers reorder and schedule operations in the input circuit so that they can run in parallel (Section 2.3.1). However, such scheduling is challenging because operations related to magic state preparation are probabilistic and effectively have non-deterministic durations. When a magic state distillation fails, the system retries the distillation until it succeeds; thus, it is a *repeat-until-success* (RUS) operation. During these repetitions, logical operations that depend on the magic state are halted, meaning that only the logical identity operations with error correction are executed during this time. The impact of distillation failures is amplified when logical operations are highly parallelized. We can rephrase this using Amdahl’s law [103], which states that the performance improvement gained by parallelization is limited by the sequential portion of the computation. If the failure case is poorly parallelized, this limits overall performance. This is an urgent issue in recent trends of optimizing magic state preparation since spacetime overhead must be reduced in the early FTQC era by aggressively employing postselection techniques that increase failure probabilities [27, 38, 99]. Therefore, to maximize the computational capability of surface-code-based FTQCs, it is necessary to explore how expensive and probabilistic operations such as magic state distillation should be handled in parallelized processes.

In this chapter, we propose a scheme to mitigate the negative performance impact caused by magic state distillation failures: specifically, we maintain a pool of magic states. We consider a surface-code-based FTQC on a 2D array of qubits and verify the magnitude of the negative impact and the effectiveness of the mitigation approach. As concrete examples, we ran two types of simulations. The first is the random quantum circuit, which is inspired by random quantum circuit sampling [8, 85] (Section 2.5.1). The second is the distributed SELECT operation [81, 84] (Section 2.5.2), which is an essential component of key quantum algorithms such as phase estimation [82] and quantum singular

value transformation [83] used in a wide variety of applications including quantum chemistry [86].

Our contributions are summarized as follows:

- We show that the impact of magic state distillation failures is amplified by parallel processing, and we quantify this impact. Specifically, in the random quantum circuit simulation with a distillation failure probability of 5%, the runtime delay caused by distillation failures is comparable to increasing the distillation time cost by over 100%. In the Dist-SELECT simulation, the runtime delay is comparable to increasing the distillation time cost by over 50% while the level-1 magic state distillation failure probability is approximately 5%.
- We introduce pools of magic states to mitigate the negative performance impact caused by magic state distillation failures on a parallel-processing quantum computer. The simulation results show that pooling effectively reduces the runtime delay, at the cost of modest additional spatial resources. Specifically, in the Dist-SELECT simulation, pooling eliminates much of the runtime delay, with an additional spatial cost of 9%. The results also show that pooling reduces the number of qubits required for distillation by 45% without increasing the temporal cost.

Our conclusion is that magic state distillation failures can add a large runtime delay, particularly when the circuit is highly parallelized, and system designers and programmers should account for these failures. Pooling provides an effective means to mitigate this runtime delay.

The rest of this chapter is organized as follows. In [Section 5.2](#), we discuss the execution model of parallel quantum algorithms we use in this chapter. In [Section 5.3](#), we analyze the negative performance impact of repeat-until-success operation failures. In [Section 5.4](#), we discuss how to mitigate the negative effect of magic state distillation failures, and present the design of a pool of magic states. In [Section 5.5](#), we describe the simulations we ran to evaluate the effectiveness of the mitigation approach. In [Section 5.6](#), we present our conclusion.

5.2 Parallel execution

Efficiently running a large program on a parallel-processing computer is challenging. A simple solution is to run one operation at a time, but this is not efficient. Another simple approach is to assign each operation a start time (clock) and execute them in synchronization across all physical qubits. This approach would not scale even without RUS operations, as synchronization overhead becomes enormous on large computers—for example, those with 20 million physical qubits [104]. With RUS operations, the situation becomes worse. When a RUS operation fails, the system needs to stop all operations except for the retrying RUS operation, which adds a significant time cost to the failure.

Zhang *et al.* [105] extracted a dependency graph from an input program and performed operations according to the dependency graph. This model is sufficiently flexible to handle RUS operations, and therefore we use this model in this chapter with two remarks.

The first concerns routing. As described in [Section 2.3](#), our target computer requires qubit mapping and routing at compile-time. These introduce additional dependencies among operations

and substantially enlarge the dependency graph. For example, if we have a CX operation on two qubits separated by 100 logical qubits, a routing path between them is required, introducing no fewer than 100 additional nodes into the dependency graph.

The other concerns retrying. Retrying a RUS operation is complicated because, although time normally flows monotonically, it effectively rolls back for the qubits involved (including routing qubits) when a RUS operation fails. The situation becomes more complicated when we consider nested failures, which is possible in two-level magic state distillation. We avoid this problem by exempting magic state distillation from this model. A magic state factory continuously runs distillation without looking at the dependency graph. At the scheduling time, the scheduler distinguishes each distillation and allocates a distillation to a magic state request. By contrast, at runtime, a consumer does not distinguish each distillation, and uses the earliest available magic state generated by the assigned magic state factory while conforming to the explicit and implicit dependencies. This also solves the nesting problem. This treatment is justified because magic state distillation has no input, and we assume that magic state factories are statically placed and that their internal routing does not interfere with external routing. The central importance of magic state distillation further justifies this special treatment.

5.3 Impact of RUS failures

In this section, we demonstrate that parallel processing amplifies the impact of RUS failures, highlighting the need for mitigation in high-performance FTQCs. In the following, we describe our motivation, formulate the problem to be addressed, and review related studies. We begin by analytically evaluating the impact of failures in sequential and parallel executions of RUS operations. Let U be a RUS operation with a failure probability p . The expected execution time required for n sequential U operations is:

$$n \sum_{i=0}^{\infty} p^i = \frac{n}{1-p}.$$

Let a_n denote the expected execution time required for n parallel U operations. It satisfies $a_0 = 0$ and $a_n = 1 + \sum_{i=0}^n \binom{n}{i} (1-p)^{n-i} p^i a_i$.

Table 5.1: Expected execution time for $p = 0.01$.

n	sequential (increase)	parallel (increase)
1	1.0101 (1.01%)	1.0101 (1.01%)
10	10.101 (1.01%)	1.0967 (9.67%)
100	101.01 (1.01%)	1.6440 (64.4%)

Table 5.1 shows the expected execution time for n sequential and parallel U operations with $p = 0.01$ and the relative increase over the execution time with $p = 0$. Failures have little impact on sequential execution but a much greater impact on parallel execution. This is because, by exploiting parallelism, parallel execution is more efficient than sequential execution, whereas the failure case is poorly parallelized.

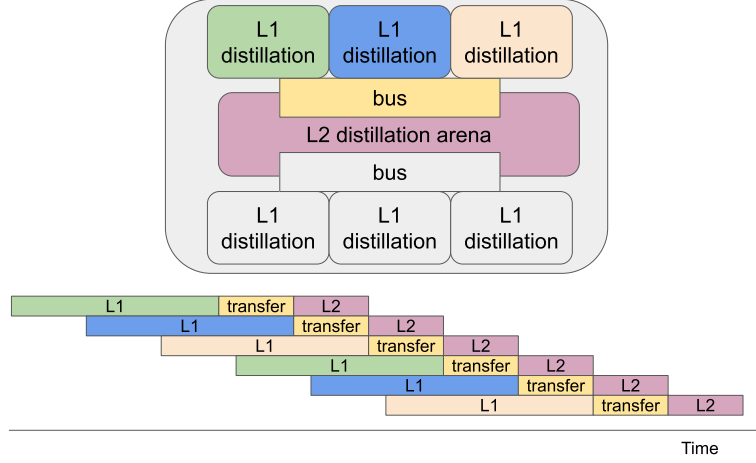


Figure 5.1: Block diagram for a two-level 15-to-1 distillation protocol (top) and one of its pipelines (bottom). Each colored box in the pipeline is a process hosted by the same-color sub-circuit in the block diagram. “L1” represents L1 distillation, “transfer” represents the L1 magic state transfer process, and “L2” represents the part of L2 distillation using the L1 magic state.

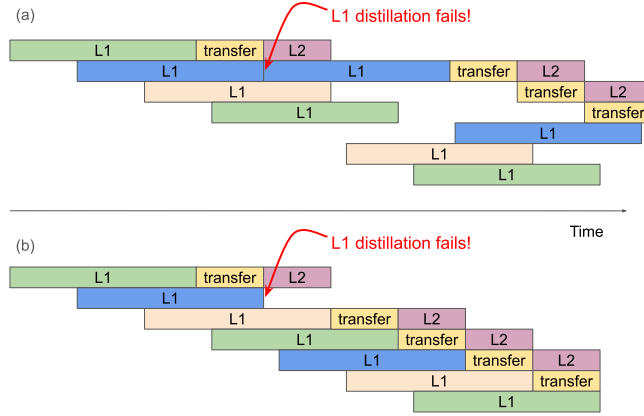


Figure 5.2: Pipeline with an L1 distillation failure.

A question that arises is whether the 64% increase in the parallel case matters given that the parallel case is much more efficient than the sequential case. It is important to note that failures can limit the benefit of parallelization, even when the failure probability is small: with $p = 0$, 100-way parallelization yields a $100\times$ speedup, but with a 1% failure probability, it achieves *only* $61\times$ speedup. In this regard, our baseline is the parallel case with $p = 0$ rather than the sequential case.

Pipelining is an effective technique for using hardware resources efficiently by capitalizing on parallelism. The magic state distillation protocols described in [Section 2.4.1](#) and [Section 2.4.2](#) leverage pipelining to minimize the distillation time cost. Consider, for example, a two-level 15-to-1 distillation factory with $n_{L1} = 6$. As shown in [Figure 5.1](#) (top), the factory consists of three components, (1) n_{L1} level-1 distillation blocks; (2) one level-2 distillation arena; and (3) two buses between the level-1 distillation blocks and the level-2 distillation arena. [Figure 5.1](#) (bottom) shows one of two pipelines in such a circuit. The impact of level-1 distillation failures could be amplified by pipelining, as illustrated in [Figure 5.2](#) (a). The longer the pipeline, the greater the cost of failures.

In this section, we have presented two examples that contain RUS operations. In both cases,

parallelization plays a key role in shortening the execution time, but it also amplifies the negative effect of RUS failures. This scenario is expected to occur generically in highly parallelized quantum programs, and it requires attention because large numbers of magic states are consumed by FTQCs.

The spatial and time costs of magic state distillation have been studied extensively [24–27, 29, 38, 90, 99]. On the magic state consumption side, many approaches have been proposed that aim for low T depth, that is, the number of layers of T gates that cannot be performed in parallel [84, 100–102]. These studies exhibit the desire to consume magic states in parallel.

Ref. [106] proposed placing a pool of magic states in each distillation factory to mitigate magic state injection failures. Its main target is photonic FBQC architectures, which are different from our target architectures. Ref. [33] proposed placing excessive L1 distillation factories in each L2 distillation factory to mitigate L1 magic state distillation failures. In Section 5.4, we present our solution, which is placing an external pool of magic states next to each distillation factory.

For two-level distillation protocols, we can mitigate the effect of L1 distillation failures relatively easily. When an L1 distillation fails, instead of waiting for the original L1 distillation factory, we can choose the best L1 distillation factory in the pipeline at runtime, as shown in Figure 5.2 (b). This mitigation approach is possible because the L1 distillation factories generate identical (albeit noisy) magic states, and the concrete circuit layout allows the runtime factory switching. Note that this approach does not work in general scenarios. As discussed in Section 5.2, routing and the computational dependency graph are intertwined, and the approach incurs a prohibitively high classical computational cost in large circuits. The purpose of this chapter is to find a mitigation approach applicable in more general scenarios.

To fully validate our statement and the solutions described in Section 5.4, an FTQC program, FTQC hardware, and FTQC compiler are needed. Many FTQC algorithms exist; however, neither FTQC hardware nor FTQC compilers are available at present. Hence, we rely on the simulations explained in Section 5.5.

5.4 Dealing with distillation failures

In this section, we discuss an approach to mitigate the negative effect of magic state distillation failures. We consider two metrics: the spatial cost (number of qubits) and the time cost (number of cycles). We refer to the time cost known at the scheduling phase as the *scheduled time cost*. Magic state distillation failures increase the time cost at run time; hence, we call this increase the *runtime delay*. The goal of the approach is to reduce the total time cost while minimizing the additional spatial cost.

We introduce metrics to analyze the impact of magic state distillation failures. Consider a computational task employing a distillation protocol with time cost D ; the task’s scheduled time cost and average runtime delay are denoted by S and R . The *relative runtime delay* is defined as $\frac{R}{S}$, representing the runtime delay relative to the scheduled time cost. Although this quantifies the impact of distillation failures, we often wish to express that impact relative to the distillation time cost. Assume that using another distillation protocol with distillation time cost E results in a scheduled time cost of $S + R$ for the same task. Let the *effective distillation time cost* of the original

distillation protocol be E^1 . This can be understood as the distillation time cost with the runtime delay incorporated. The original distillation protocol and a hypothetical distillation protocol with time cost E and no failure probability lead to the same total time cost. We further define the *relative distillation time cost increase* as $\frac{E-D}{D}$. For example, let us suppose the original distillation time cost is 100, the scheduled time cost for the original distillation time cost is 1200, the runtime delay is 300, and the scheduled time cost for the distillation time cost 200 is 1500. Then the relative runtime delay is 25% and the relative distillation time cost increase is 100%. In this case, the runtime delay effectively doubles the distillation time cost.

To hide the latency caused by distillation failures, we introduce a pool of magic states, which is a first-in-first-out queue, to each magic state factory. We expect the pool to serve as a buffer between the factory and the consumers of magic states. Each factory continuously distills magic states, and the distilled magic states are stored in the pool. On the consumption side, upon the arrival of a magic state request, the pooled magic state is transferred from the pool to the consumer, if possible; otherwise, the request needs to wait for the factory. Note that this behavior is determined at runtime, rather than at the scheduling phase, similar to magic state distillation itself. This aligns with the execution model discussion in Section 5.2. The additional spatial cost is the number of qubits required for the pool. In a pool, using a smaller code distance than that used for usual data qubits may be acceptable in terms of the error probability because magic states remain there for a relatively short period of time.

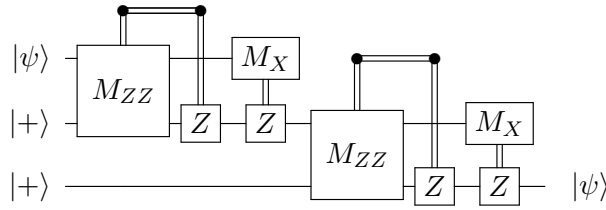


Figure 5.3: Transferring a quantum state. M_{ZZ} denotes the ZZ measurement and M_X denotes the X measurement.

We transfer a quantum state across qubits with quantum teleportation, i.e., entangling the qubits and measuring the source qubit. Entangling adjacent qubits is achieved by lattice surgery [41] (Section 2.2.5) and hence entangling two qubits can be naively performed in $(l+1)d$ cycles, where l is the length of the route between the two qubits and d is the code distance, as depicted in Figure 5.3. Note that Z gates in Figure 5.3 have no time cost because we can execute them classically, using the Pauli feedforward mechanism [59]. When the route is linearly shaped, though, the $(l+1)$ lattice surgery operations can be carried out simultaneously and the entanglement can be established in d cycles. Hence, unlike pools on classical computers, this pool introduces no additional latency. The benefit of parallelization grows with queue length. Figure 5.4 shows typical transfer scenarios and Figure 5.5 is a state transition diagram of a single-entry pool.

¹The scheduled time cost tends to increase as the distillation time cost increases, but it is not a strictly increasing function. Therefore, multiple E values may exist; in such a case, we select the largest value.

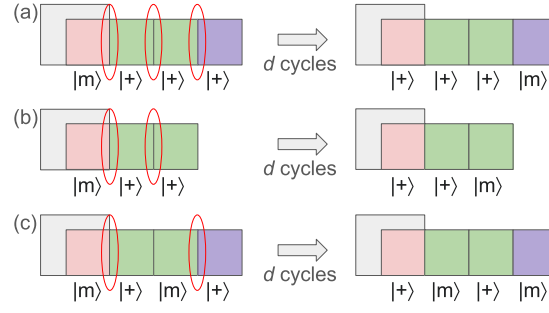


Figure 5.4: Transferring quantum states across a pool with two entries. Different scenarios presented in rows demonstrating the status of qubits before (left) and after (right) the transfer. Pink, green, and purple squares are qubits in magic state factories, pools, and the consumers of magic states, respectively. Each red ellipse is a lattice surgery operation, and $|m\rangle$ is a magic state. The routing qubits between the pool and the consumer are omitted. (a) The pool is empty and the magic state is transferred directly to the consumer. (b) There is no magic state request and the magic state is stored in the pool. (c) A pooled magic state is transferred to the consumer while another magic state is stored in the pool.

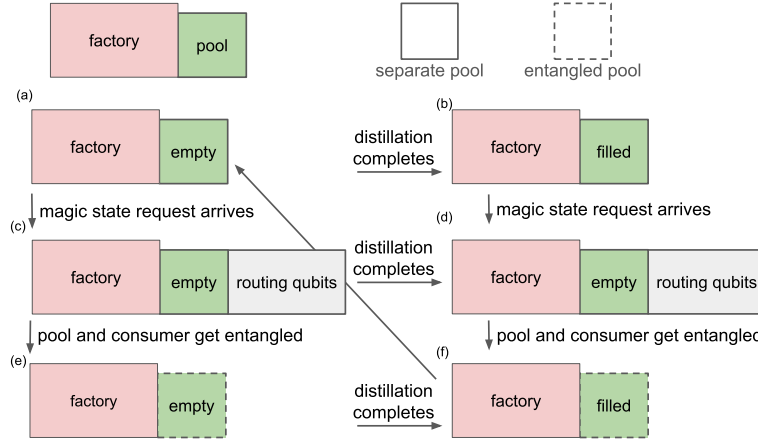


Figure 5.5: Runtime behavior of pooling illustrated as a state diagram. A pool is entangled with the consumer if it has dashed boundaries. (a)–(f) represent a factory with (a) an empty pool, (b) a filled pool, (c) an empty pool being entangled with the consumer (through the routing qubits), (d) a filled pool being entangled with the consumer (through the routing qubits), (e) an empty pool entangled with the consumer, waiting for a magic state, and (f) a filled pool entangled with the consumer. At (f), the consumer can use the magic state and the state immediately transitions to (a).

This analysis assumes the presence of a route between the endpoint qubits, which implies that the intermediate qubits do not store magic states in our context. As a consequence, transferring quantum states in a denser pool would be more expensive. Assuming that we distill and consume a magic state every nd cycles, a pool with nm entries can retain $\frac{(n-1)m}{n}$ states on average, as depicted in Figure 5.6. The sliding scheme discussed in Ref. [60] may accelerate the transfer process when the pool is densely occupied.

The speeds of magic state distillation and consumption must be well balanced to achieve optimal performance. Ideally, the distillation speed should slightly outpace the consumption speed. If consumption is faster than distillation, it will deplete the pool and the consumer of magic states

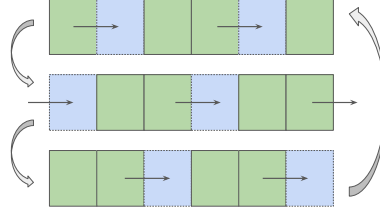


Figure 5.6: Pool with six entries connected to a magic state distillation factory that generates a magic state for every $3d$ cycles. A blue square represents an empty qubit and a green square represents a filled qubit. Each row represents the pool in a different time slice. A state is enqueued to the left end and dequeued from the right end.

will be blocked, thereby extending the execution time. If the distillation speed exactly matches the consumption speed, an empty pool will never be replenished. Finally, if the distillation speed is excessively rapid, surplus magic states are discarded.

5.5 Performance evaluation

In this section, we conduct two types of simulations. In the first, we simulate a random quantum circuit to evaluate the impact of magic state distillation failures under various failure probabilities. The second, a more practical simulation, models the distributed SELECT circuit described in [Section 2.5.2](#). In addition to evaluating the impact of distillation failures, this simulation primarily aims to assess the effectiveness of the mitigation approach discussed in [Section 5.4](#).

5.5.1 Random quantum circuit simulation

We conduct a simulation of a randomly generated circuit inspired by random quantum circuit sampling ([Section 2.5.1](#)). In the simulation, we apply “layers” to a plane of logical qubits. Each layer consists of single-qubit and two-qubit operators, the latter of which we call *couplings*. [Table 5.2](#) shows the parameters.

Table 5.2: Random quantum circuit simulation parameters

symbol	description
d	The code distance of the surface code.
W	The width of the qubit plane.
H	The height of the qubit plane.
L	The number of layers to be applied.
D	The coupling distance.
p_{fail}	The failure probability of magic state distillation.

For each layer, we apply a randomly chosen single-qubit operator (S , H , or T) to each qubit and then apply as many couplings as possible, with the following constraints:

- Each qubit is involved in at most one coupling.
- Each coupling’s L1 distance is less than or equal to the coupling distance parameter D .

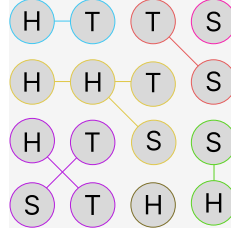


Figure 5.7: Layer of a random quantum circuit with $W = H = 4$, and $D = 2$. Each circle represents a qubit and the character inside represents a single-qubit operator. Each line that connects two circles represents a coupling operator. Couplings that belong to the same coupling group have the same color, as do the qubits involved in such couplings.

Figure 5.7 shows a layer of a random quantum circuit.

Table 5.3: Time costs of operators

operator	H	S	T	coupling
cost	$3d$	$2d$	$3d NB(1, 1 - p_{\text{fail}}) + 4d$	$2d$

Table 5.3 shows the time cost of operators in the simulation. Using lattice surgery, we can implement H and S gates with $3d$ and $2d$ cycles, respectively. The time cost of a T gate depends on the distillation protocol, but in our simulation, we simply assume distillation behaves as a Bernoulli trial with success probability $1 - p_{\text{fail}}$ and each trial takes $3d$ cycles. $NB(r, p)$ in the table stands for the negative binomial distribution and $3d NB(1, 1 - p_{\text{fail}}) + 3d$ models the time cost of a distillation. An additional d cycles are required for gate teleportation. Note that we do not count the time cost of an S fix-up, required with probability $1/2$, because it can be performed with zero time cost and additional ancilla qubits. We assume that a coupling operator is a simple two-qubit operator, such as CX . Such an operator runs in $2d$ cycles, even when the endpoints are distant, assuming a route exists between them.

Our simulation implicitly manages routing and distillation, following the principles laid out in Section 2.3 and Section 5.2. To account for the complexity of routing, we introduce the concept of a *coupling group*, which is a collection of couplings. Two couplings belong to the same coupling group when they intersect with each other as line segments. In Figure 5.7, couplings that belong to the same coupling group have the same color. Algorithm 4 is the simulation algorithm. As in the algorithm, single-qubit operators can run in parallel, whereas a coupling is bottlenecked by the slowest qubit among those involved in the same coupling group. $elapsed[q]$ in the algorithm represents the elapsed cycles on qubit q . The circuit's elapsed cycles are represented by the maximum elapsed value across all qubits.

We set $W = H = 64$ and $L = 100^2$. Figure 5.8 and Figure 5.9 show the relative runtime delay and relative distillation time cost increase for various D and p_{fail} settings, respectively. The range of p_{fail} includes the range of the failure probabilities of single-level 15-to-1 distillation protocols in Ref. [27]. For example, the failure probabilities of the protocols with the physical error probability

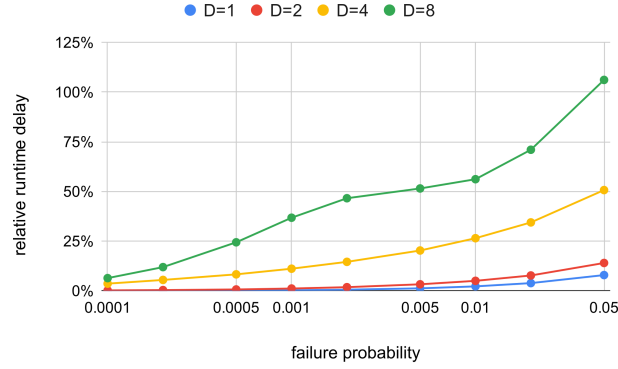
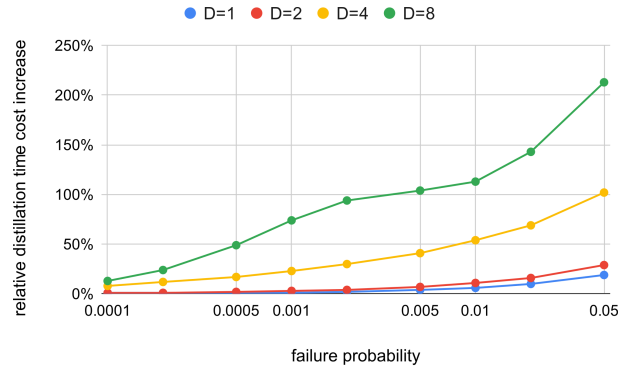
²Because every time cost is proportional to d , the relative runtime delay and relative distillation time cost increase are independent of d .

Algorithm 4 Random quantum circuit simulation

```

1: for layer in all the layers do
2:   for (q, gate) in single-qubit gates at layer do
3:     elapsed[q]  $\leftarrow$  elapsed[q] + the time cost of gate
4:   end for
5:   for group in coupling groups at layer do
6:     m  $\leftarrow$  0
7:     for (q1, q2) in group do
8:       m  $\leftarrow$  max(m, elapsed[q1], elapsed[q2])
9:     end for
10:    for (q1, q2) in group do
11:      elapsed[q1]  $\leftarrow$  m + 2d
12:      elapsed[q2]  $\leftarrow$  m + 2d
13:    end for
14:  end for
15: end for

```

**Figure 5.8:** Relative runtime delay for each parameter setting.**Figure 5.9:** Relative distillation time cost increase for each parameter setting.

10^{-4} and 10^{-3} are close to 0.001 and 0.05, respectively. As described above, the coupling distance models coupling locality: the smaller the coupling distance, the more localized the couplings. The figures show that even moderate coupling distances and failure probabilities can incur significant additional time costs.

5.5.2 Dist-SELECT simulation: scheduling

We simulate a system consisting of M -multiplexed SELECT sub-circuits (Section 2.5.2) that share the Pauli target qubits, forming the core part of the Dist-SELECT circuit. In the simulation, we assume that we can run at most P controlled Pauli operators in parallel on the Pauli target qubits. P reflects the routing efficiency that depends on the quantum computer architecture and compiler. This is the only part of our simulation that takes the routing efficiency into account explicitly. Table 5.4 shows the parameters.

Table 5.4: Dist-SELECT simulation parameters

symbol	description
d	The code distance of the surface code.
d_{pool}	The code distance for the magic state pool.
$\#_{\text{MSD}}$	The number of magic state factories.
M	Dist-SELECT parallelism.
P	The number of controlled Pauli operators that can run in parallel on the Pauli target qubits.
N	The number of the Pauli target qubits.
L	The original length of the SELECT operation.
D	The distillation cycles used during scheduling.
p_{phys}	The physical error probability.

Scheduling gates in the SELECT circuit is relatively straightforward because it has a clear critical path and we mostly ignore the topological aspects. We schedule the operations greedily, allocating two special resources: magic states and controlled Pauli operators on the Pauli target qubits. Algorithm 5 shows the scheduling algorithm. “Operation” in the algorithm represents a basic operation, such as a Pauli or measurement operation, accompanied by dependency information. Scheduling M -multiplexed SELECT sub-circuits that share the Pauli target qubits can be implemented by adding small extensions to Algorithm 5.

Our magic state factory relies heavily on pipelining, and to work efficiently, it needs to store partially completed calculations performed in the previous round of distillation. However, in the factory we use smaller code distances, which provide less protection against errors. This means states in the factory decay, and we cannot just keep them indefinitely. Hence, our magic state factories continuously run, and there is no on-demand distillation. The scheduler expects a magic state factory to generate a magic state every D cycles, and the scheduler’s role is to select the right distillation round for each magic state request. This is aligned with the execution model discussion in Section 5.2.

Figure 5.10 shows the magic state distillation scheduling process for a scenario with a single factory. In Figure 5.10, the consumer qubit remains blocked until c_1 . The scheduler looks at the factory’s schedule to minimize the duration of the blockage at the consumer qubit of the magic state. In the example case, distillation #2 is the best choice because choosing distillation #3 would block

Algorithm 5 *Schedule operation*

```

1: if operation has already been scheduled then
2:   return
3: end if
4:  $clock \leftarrow 0$ 
5: for ancestor in the ancestors of operation do
6:   Schedule ancestor.
7:    $s \leftarrow$  ancestor's scheduled clock
8:    $cycles \leftarrow$  ancestor's time cost
9:    $clock \leftarrow \max(clock, s + cycles)$ 
10: end for
11: if operation works on the Pauli target qubits then
12:   Schedule the operation on the Pauli target qubits, and let  $c$  be the scheduled clock.
13:    $clock \leftarrow c$ 
14: else if operation is a magic state preparation then
15:   Request a magic state from the factories, and let  $c$  be the scheduled clock.
16:    $clock \leftarrow c$ 
17: end if
18:  $clock$  is the scheduled clock for operation

```

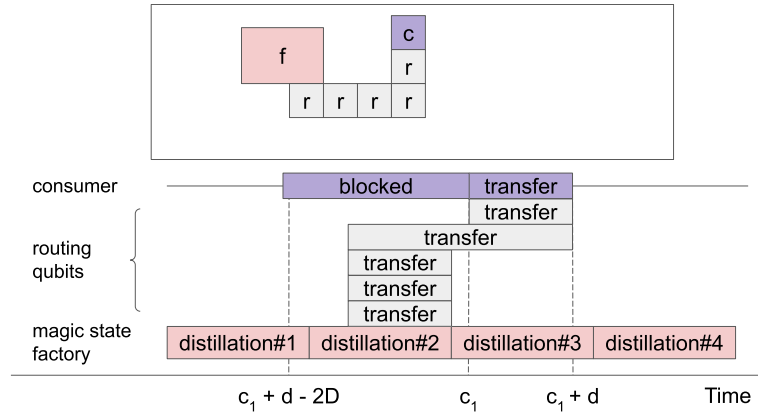


Figure 5.10: Top: Block diagram depicting a magic state distillation factory (pink), the consumer (purple), and the routing qubits (gray) in the circuit. Bottom: Schedule for each component.

the consumer. We call the magic state transfer operation at the consumer (the purple “transfer” block in the figure) *magic state preparation*. In our simulation, routing qubits are treated implicitly and we assume that there are sufficient routing qubits around a distillation factory to avoid congestion. Hence, despite the fact that the magic state preparation operation and distillation #3 are performed simultaneously, we assume that there are no conflicts between them and the scheduler can assign distillation #3 to another magic state request. We can easily expand this single-factory logic to the multi-factory case by selecting the best-matching distillation round in all the factories.

If distillation #2 has already been assigned to another magic state request in Figure 5.10, the scheduler needs to look for another one. Our natural choice would be distillation #1, but this means that the magic state would be stored in the routing qubits temporarily. Storing states in routing qubits for an extended period leads to routing congestion, which our simulation cannot model, as

Algorithm 6 Execute

```

1: for operation in all the operations do
2:   for i in  $0..M$  do
3:     push(queue, (operation, i))
4:   end for
5: end for
6: while queue is not empty do
7:   (op, index)  $\leftarrow$  pop(queue)
8:   if (op, index) is complete then
9:     continue
10:  else if (op, index) has an incomplete ancestor then
11:    continue
12:  end if
13:  Mark (op, index) as complete.
14:  Let q be the qubit on which (op, index) runs.
15:  Propagate delay information based on dependencies.
16:  if op is a magic state preparation then
17:    Let d be the delay caused by the magic state distillation based on the given distribution.
18:    delay[q]  $\leftarrow$  delay[q] + d
19:  end if
20:  for (n, i) in the descendants of (op, index) do
21:    push(queue, (n, i))
22:  end for
23: end while

```

discussed above. To avoid this scenario, we impose the following rule: The assigned distillation must not precede the magic state preparation operation by more than $2D - d$ cycles. Hence the scheduler will choose distillation #3.

5.5.3 Dist-SELECT simulation: execution

Once scheduling is complete, we simulate the execution phase. The simulation calculates *delay*, the number of cycles delayed relative to the schedule for each logical qubit (the Pauli target qubits are treated as one qubit). It processes operations according to the operation dependencies while propagating delay information. Algorithm 6 shows the algorithm.

Delays caused by magic state distillation failures are amplified by the delay propagation process, which is defined by the following criteria.

- Propagate a delay between two operations when they have a dependency.
- A magic state request waits for all preceding requests to the associated magic state factory.

In particular, all controlled Pauli operators on the Pauli target qubits share the Pauli target qubits and have an implicit dependency on each other. Hence, once a delay is propagated to the Pauli target qubits, the delay affects all the operations running on them thereafter. This reflects the routing difficulty on the Pauli target qubits.

Consider a magic state preparation operation o on qubit q . The following procedure calculates the delay caused by magic state distillation. Let c be the scheduled start time of o , and let c_d be the scheduled start time of the distillation associated with o . As shown in Figure 5.10, $c_d \geq c + d - 2D$. Let $delay$ be the delay on q just before o starts, and e be the first distillation completion that satisfies the following:

- Prior magic state requests to the factory have already been completed at the beginning of the distillation.
- $e \geq c_d + delay + D$.

The first condition states that magic state requests are processed in order, and the second condition states that the runtime behavior should not overtake the schedule. o is completed at $\max(c + delay + d, e)$ and the new delay on q becomes $\max(c + delay + d, e) - c - d$. This behavior generalizes when pooling is enabled, as discussed in Section 5.4.

To reduce the simulation time cost, we separate the magic state distillation process from the main simulation. Specifically, we precompute the distribution of distillation time costs. In the main simulation, each distillation time is treated as a random variable drawn from this distribution.

5.5.4 Dist-SELECT simulation: parameter settings

In this subsection, we describe the parameter settings (see Table 5.4 for the list of parameters). We set some parameters directly from our assumptions and derive others accordingly. We choose the problem size as $N = 2^{12}$ and $L = 2^{24}$. P and M control the parallelization level. We set $P = 32$ and $M = 128$ to simulate highly parallelized circuits. p_{phys} denotes the physical error probability, and we choose a realistic value, 10^{-3} . We additionally make the following assumptions:

- We want the error probability of the Dist-SELECT circuit to be below 1%.
- We use the $|CCZ\rangle$ distillation protocols in Ref. [27].
- The error probability of a logical qubit per cycle is $p_L(d) := 0.1(100p_{\text{phys}})^{\frac{d+1}{2}}$.

To maintain the overall error probability below 1%, we want the error probabilities of magic states and other components (Clifford operations) to be under 0.5%. Because we need L CCZ states in the circuit, the $(15\text{-to-}1)_{13,5,5}^4 \times (8\text{-to-CCZ})_{23,13,15}$ protocol—whose error probability per magic state is 1.8×10^{-10} —is suitable. Hence, we set $D = 60$.

To determine the code distance, we first need to make a rough estimate of the spatial and time cost. We allocate four logical qubits for each Pauli target qubit to support the parallel execution of gates, and two logical qubits for each index and ancilla qubit. Assuming the spatial cost of temporary ancilla qubits is negligible, the total number of logical qubits—excluding those used in magic state distillation factories—is $4N + 2 \times 2M \log(L) \approx 2.9 \times 10^4$. The product of $p_L(d)$, spatial cost, and time cost must be below 0.005; hence, we choose $d = 27$. We perform simulations with various $\#_{\text{MSD}}$ values in the range of $[60, 160]$. 60 factories use approximately 5% and 160

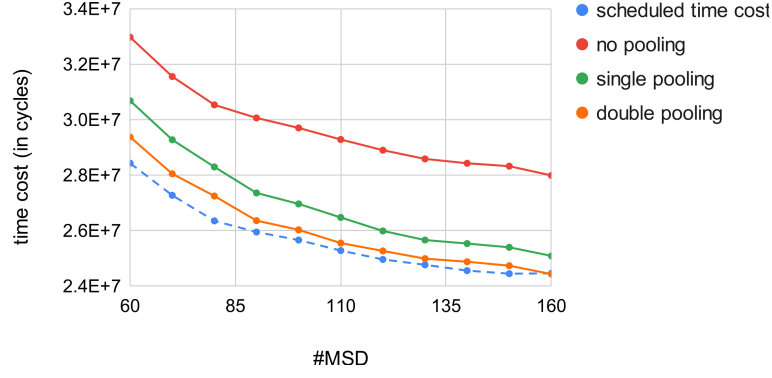


Figure 5.11: Pooling reduces the runtime delay. The blue dotted line shows the scheduled time cost and the other lines represent the total time cost including the runtime delay.

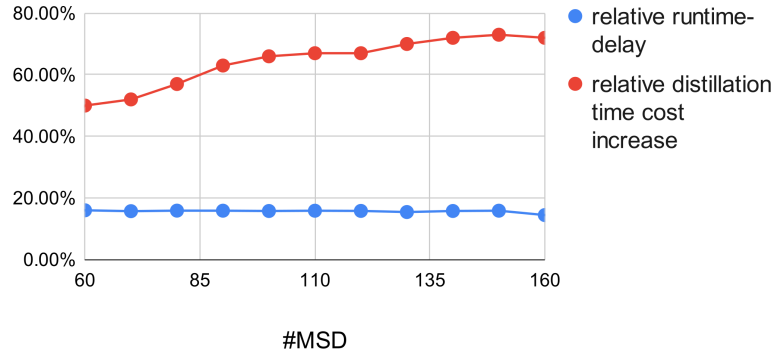


Figure 5.12: Impact of distillation failures.

factories use approximately 12% of the total qubits, both of which are reasonable. Our simulations show that more factories would not result in a substantial reduction in the time cost.

As discussed in Section 5.4, d_{pool} , which is the code distance used in the pool, can be smaller than d . Let E be the expected value of the distillation cycles. Then d_{pool} must satisfy

$$L(3(E + d)p_L(d_{\text{pool}}) + 1.8 \times 10^{-10}) < 0.005.$$

$d_{\text{pool}} = 23$ is the smallest odd integer satisfying this inequality. Given that one factory uses 34,052 physical qubits, the spatial overhead of the pool is $\frac{3 \times 2d_{\text{pool}}^2}{34052} \approx 9\%$ of the factory size.

5.5.5 Dist-SELECT simulation: results

Figure 5.11 shows the scheduled time cost (blue dotted line) and total time cost (red solid line), and Figure 5.12 shows the relative runtime delay and relative distillation time cost increase for each $\#_{\text{MSD}}$ value. As shown in Figure 5.12, the impact of magic state distillation failures is substantial: For most $\#_{\text{MSD}}$ values, the relative distillation time cost increase is over 50%. This also shows that adding factories reduces the scheduled time cost and hence the total time cost, whereas the relative runtime delay is almost constant.

The impact of pooling on runtime delay is demonstrated in Figure 5.11. Pooling with a single entry (“single pooling”) effectively reduces the runtime delay, with reductions ranging from 50% to

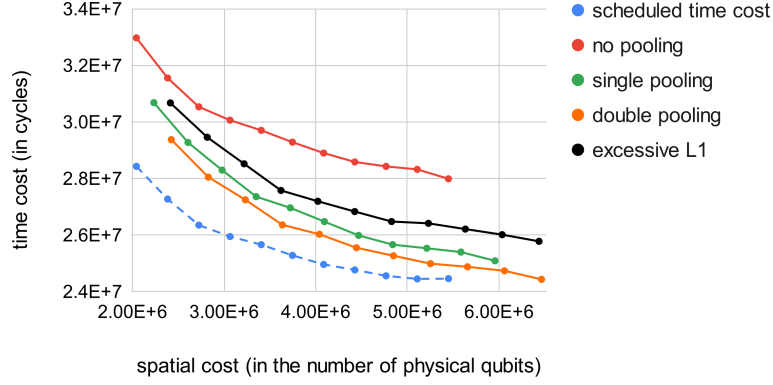


Figure 5.13: Spatial and time cost trade-off with pooling.

82%. Pooling with two entries (“double pooling”) further reduces the runtime delay. To find the optimal balance between the magic state distillation and consumption speeds for pooling, we ran the simulation with various magic state consumption speeds and selected the value that had the lowest time cost for each $\#_{\text{MSD}}$ setting. Note that in the figure we ignore the spatial overhead of pooling. We consider it in Figure 5.13, where the horizontal axis represents the number of physical qubits required for magic state distillation and pooling, instead of $\#_{\text{MSD}}$. Evidently, the configurations with pooling present better trade-offs than the configuration without pooling. Ref. [33] proposed having excessive L1 distillation factories in each L2 distillation factory to mitigate L1 distillation failures, and the black line in Figure 5.13 (“excessive L1”) corresponds to the solution. The figure shows that pooling is better than having excessive L1 factories.

We can also use pooling to reduce the spatial cost. In Figure 5.13, $\#_{\text{MSD}} = 150$ without pooling uses 5,107,800 qubits and the total time cost is 28,313,114 cycles. $\#_{\text{MSD}} = 70$ with double pooling uses 2,828,000 qubits and the total time cost is 28,038,996 cycles. Hence, pooling offers a spatial cost reduction of 45% with a smaller time cost.

The spatial cost of additional magic state factories might be underestimated because when we add a magic state factory, we usually need to add some routing qubits, which are not counted in our assessment. By contrast, no additional routing qubits are required for pooling. Hence, pooling may be more competitive than Figure 5.13 shows.

5.6 Conclusion

In this chapter, we showed that the performance impact of magic state distillation failures can be substantial on a parallel-processing quantum computer. To mitigate this effect, we proposed a mitigation scheme based on a pool of magic states. To evaluate the magnitude of this impact and the effectiveness of the mitigation approach, we conducted numerical simulations using random quantum circuits and the Dist-SELECT circuit.

In simulations of random quantum circuits, we observed a relative distillation time cost increase exceeding 100% under modest failure probabilities and limited locality. For the Dist-SELECT circuit, we observed a relative distillation time cost increase exceeding 50%. Hence, we conclude that magic state distillation failures can introduce significant runtime delays, particularly in highly

parallelized circuits; system designers need to pay attention to such failures. Although the actual numbers depend on the computer architecture and the target algorithm, we believe that the overall conclusion is robust.

For the Dist-SELECT circuit, pooling reduces most of the runtime delay while incurring only a small additional distillation spatial cost. Pooling offers better space-time trade-offs compared with configurations without it. We also observed a 45% distillation spatial cost reduction without increasing the time cost. Hence, we conclude that pooling effectively mitigates the performance degradation caused by magic state distillation failures. Moreover, we observed that pooling outperforms the approach of equipping each L2 distillation factory with an excessive number of L1 factories [33].

The scheduler described in Section 5.5 is simple. More sophisticated schedulers should be able to exploit parallelism further to shorten the program execution time. For example, commuting operations on the same qubits can run in parallel with extra space overhead [26]. With such sophisticated schedulers, the effect of magic state distillation failures can increase further, underscoring the importance of mitigating their effects.

As discussed in Section 5.1, recent magic state distillation proposals [29, 38, 99] offer reduced overheads but result in greater failure probabilities. With such schemes, mitigating the effects of distillation failures becomes more important.

We have focused on magic state distillation primarily because it is a frequently used RUS operation. We believe that any frequently used RUS operation could lead to similar effects, though the mitigation strategies may differ from those proposed here. For example, RUS operations are extensively utilized in Ref. [34]. State injection used by magic state distillation is another example of a probabilistic procedure [107–109]. While they are not distillation processes, failures in these procedures may similarly affect the computation time.

A further relevant example is entanglement distillation. A maximally entangled state $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ plays a crucial role in distributed quantum computation. Entanglement distillation [110] is a process to prepare a state close to $|\Phi\rangle$ in a distributed environment using only local operations and classical communication, given noisy states as input. Given its importance, its resource costs—including its failure probability—have been well studied [111, 112]. However, we are not aware of any studies discussing the negative impact of entanglement distillation failures on the total system. Assessing this impact is crucial for the development of efficient, highly distributed systems.

Chapter 6

Locality-Aware Pauli-Based Computation for Local Magic State Preparation

Magic state distillation, a process for preparing magic states needed to implement non-Clifford gates fault-tolerantly, plays a crucial role in fault-tolerant quantum computation. Historically, it has been a major bottleneck, leading to the pursuit of computation schemes optimized for slow magic state preparation. Recent advances in magic state distillation have significantly reduced the overhead, enabling the simultaneous preparation of many magic states. However, the magic state transfer cost prevents the conventional layout from efficiently utilizing these magic states, highlighting the need for an alternative scheme optimized for highly parallel quantum algorithms. In this chapter, we propose locality-aware Pauli-based computation, a novel compilation scheme that distills magic states in the computation area, aiming to reduce execution time by minimizing the magic state transfer cost and improving locality. Numerical experiments on random circuit sampling and 2D Ising Hamiltonian simulation demonstrate that our scheme significantly reduces execution time—while incurring little or no additional spatial overhead—compared to sequential Pauli-based computation, a conventional computation scheme, and scales favorably with increasing qubit count. This chapter is based on “**Locality-Aware Pauli-Based Computation for Local Magic State Preparation**,” Hirano, Fujii, *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 670–680 [52].

6.1 Introduction

Quantum computers promise to solve problems of practical importance that are believed to be intractable for classical computers—for example, simulating quantum physics [4] and integer factoring [3]. Existing quantum computers, often referred to as noisy intermediate-scale quantum (NISQ) devices, are highly susceptible to hardware errors, preventing them from executing complex quantum algorithms such as Shor’s factoring algorithm [3]. To fully realize this promise, we need fault-tolerant quantum computers (FTQCs) [9] based on quantum error correction (QEC) [10, 57],

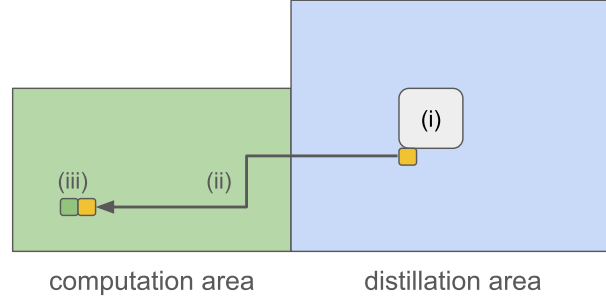


Figure 6.1: Performing a T gate in the conventional layout. (i) Magic state distillation. (ii) Magic state transfer from the distillation area (blue) to the computation area (green). (iii) Gate teleportation.

which protects information from errors by encoding *logical qubits* into many physical qubits. Surface codes [30, 69] (Section 2.2.4) are among the most promising candidates for QEC because of their relatively high error threshold and compatibility with nearest-neighbor connectivity. Experimental quantum error correction has been demonstrated using the surface code architecture on quantum devices with nearest-neighbor interactions, such as superconducting qubits [11].

To perform computations on an FTQC, we need *logical gates* applied to logical qubits. For the surface code, the Clifford+ T gate set, consisting of $\{H, S, T, CX\}$ gates, is often adopted to enable universal quantum computation. Among these, the Clifford gates $\{H, S, CX\}$ are relatively inexpensive, whereas the T gate is more resource-intensive. To perform a T gate, we prepare a *magic state*, $T|+\rangle$, through a process called *magic state distillation* [14] (Section 2.4). Magic state distillation is expensive and has long been considered a major bottleneck in fault-tolerant quantum computation. This has led to the pursuit of compilation schemes optimized for the high cost of magic state preparation, where a compilation scheme consists of the *transpilation* process, *layout*, and other compilation algorithms (Section 2.3.1).

Sequential Pauli-based computation (SPC) [26] (Section 2.3.2) is one such scheme. It transpiles a sequence of Clifford+ T gates into multi-qubit Pauli measurements. SPC uses a layout that separates the computation area from a dedicated distillation area containing multiple magic state factories. In this layout, executing a T gate involves three steps: (i) distilling a magic state in the distillation area, (ii) transferring it to a qubit adjacent to the target data qubit in the computation area, and (iii) performing gate teleportation (Figure 6.1). This layout has been widely adopted in resource estimation and compilation studies for FTQC with nearest-neighbor connectivity [33, 43–47].

Recent advances have significantly reduced the overhead of magic state distillation [23, 27, 29, 38, 49, 99], enabling the simultaneous preparation of many magic states. However, the cost of magic state transfer—which scales with the length of the transfer path—prevents the conventional layout from efficiently utilizing these magic states. This highlights the need for an alternative layout and a transpilation strategy optimized for highly parallel quantum algorithms.

In this chapter, we propose locality-aware Pauli-based computation, a novel FTQC compilation scheme optimized for highly parallel quantum algorithms. This scheme performs magic state distillation locally in the computation area, relying on the recent distillation advances that have

greatly reduced the spatial cost. To make full use of this locality, our scheme transpiles an input quantum algorithm composed of Clifford+ T gates into a sequence of single-qubit and two-qubit Pauli measurements with magic states, while preserving the gate locality of the input algorithm. The combination of gate locality and local magic state preparation enables the parallel execution of many T gates. High-performance distillation techniques rely on postselection to reduce overhead, resulting in low distillation success probabilities. We mitigate this by running multiple parallel distillation processes for each magic state. The enhanced locality offers greater parallelism than existing parallel-computing schemes based on the conventional layout [33, 43, 46]. Moreover, our scheme is compatible with many routing and scheduling algorithms developed for conventional layouts, opening the door to even more efficient compilation strategies.

To evaluate the scheme’s performance, we developed a scheduler and a runtime simulator that implement locality-aware Pauli-based computation. We performed numerical simulations of two algorithms—random circuit sampling (Section 2.5.1) and 2D Ising Hamiltonian simulation (Section 2.5.3)—using the scheduler and simulator. For 6×6 (12×12) random quantum circuit sampling, we observed a 14% (71%) reduction in execution time compared to SPC, with little or no additional spatial overhead. For 6×6 (12×12) Hamiltonian simulations, we observed a 48% (84%) reduction in execution time compared to SPC, again with little or no additional spatial overhead. These results demonstrate that locality-aware Pauli-based computation scales favorably in execution time as the number of qubits increases. Therefore, we conclude that locality-aware Pauli-based computation will become a standard compilation scheme targeting highly parallel quantum algorithms when high-performance magic state distillation is available. Our scheduler and runtime simulator reveal the relationship between the magic state distillation cost—including its success probability—and the overall computational cost for a given quantum algorithm. This provides useful design guidelines for developing magic state distillation protocols.

The rest of this chapter is organized as follows. In Section 6.2, we detail our proposal, locality-aware Pauli-based computation, including theoretical analyses and comparisons with other schemes. In Section 6.3, we conduct a performance evaluation through resource estimation with random circuit sampling and Hamiltonian simulation with Trotterization. Finally, Section 6.4 concludes the chapter with a summary of our findings.

6.2 Locality-aware Pauli-based computation

In this section, we detail our proposal, locality-aware Pauli-based computation. In Section 6.2.1, we provide an overview of the scheme without going into details. Next, we list related studies in Section 6.2.2. We then examine the validity of our assumption regarding in-place magic state distillation in Section 6.2.3. Finally, we present the quantum ISA (Section 6.2.4) and describe the transpilation process (Section 6.2.5).

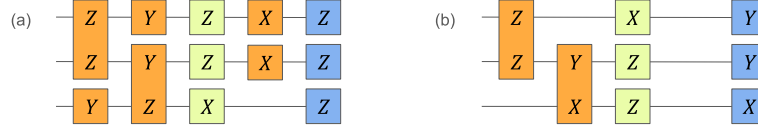


Figure 6.2: Circuit translation of locality-aware Pauli-based computation (LAPBC): (a) original circuit; (b) LAPBC translation result. Pauli gates in the circuits are omitted. See Figure 2.19 for the color coding scheme.

6.2.1 Overview of the scheme

We propose locality-aware Pauli-based computation (LAPBC), a new compilation scheme (Section 2.3) that consists of a transpilation algorithm, a qubit layout, and other compilation algorithms, all of which are designed with an emphasis on locality. We provide an overview of the transpilation algorithm and qubit layout below.

LAPBC transpiles a quantum IR program (Section 2.3.1) into a sequence of single-qubit $\frac{\pi}{8}$ Pauli rotations, single-qubit or two-qubit $\frac{\pi}{4}$ Pauli rotations, and Pauli measurements (Figure 6.2). This translation removes single-qubit Clifford gates from the original program and enables efficient program execution in a manner similar to SPC (Section 2.3.2). However, as a crucial difference from SPC, we do not propagate all Clifford gates backward; rather, we restrict this propagation to single-qubit Clifford gates. This translation preserves the support of each gate, that is, the set of logical qubits that the gate acts on nontrivially. We refer to this as the preservation of the gate locality of the original program. As a result, the proposed transpilation allows parallel gate execution.

Another key feature of LAPBC is that we adopt *factory-less layouts*, which are qubit layouts without separate magic state factories. Magic state distillation is performed using routing qubits, under the assumption that distillation factories are small, motivated by recent progress on space-efficient magic state distillation protocols [29, 38]. As discussed in Section 6.1, the conventional layout (Figure 6.1) leads to significant magic state transfer costs and is not well suited for highly parallel algorithms, for the following reasons. First, since a T gate requires a path between the computation and distillation areas, the maximum parallelism is limited by the perimeter of the computation area, which is $\mathcal{O}(\sqrt{N})$, where N is the number of logical data qubits (Figure 6.3 (left)). Second, the negative impact of distillation failures can be amplified by the conventional layout. Magic state distillation is a probabilistic process and may fail. When this happens, the system retries distillation, and any computation dependent on the magic state must wait for its completion. In the conventional layout, computations involving qubits on the magic state transfer path must also be stalled, even when they are logically independent of the magic state (Figure 6.4), thereby lowering the effective parallelism [51]. Although this issue can be mitigated, doing so introduces additional computational overhead.

A factory-less layout enables local distillation, reduces magic state transfer costs, and allows $\mathcal{O}(N)$ parallel T gate execution (Figure 6.3 (right)). Local distillation also minimizes the runtime delay caused by distillation failures. By combining the transpilation algorithm and the factory-less layout, locality-aware Pauli-based computation aims to shorten the execution time of highly parallel

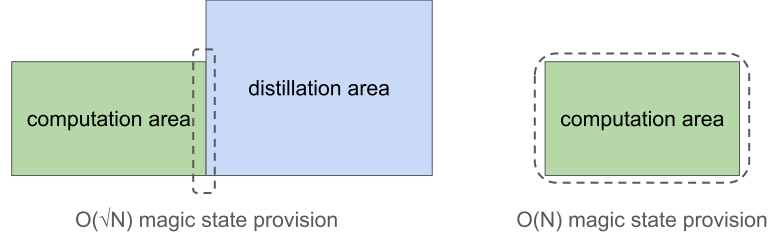


Figure 6.3: Magic state provision in the conventional layout (left) and locality-aware Pauli-based computation (right).

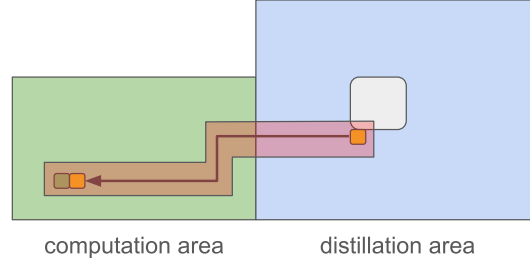


Figure 6.4: A magic state distillation failure (gray) causes a runtime delay that propagates along the magic state transfer path (red).

quantum algorithms.

6.2.2 Related studies

SPC [26] is a compilation scheme designed for the conventional layout. It efficiently performs computations by eliminating Clifford gates and executing multi-qubit Pauli measurements sequentially. While Ref. [26] includes a strategy for parallel computation, Ref. [44] reports this strategy to be inefficient in practice.

Several resource estimation and compilation studies for the conventional layout support parallel computation [33, 43–47]. The limitation on the T parallelism discussed in Section 6.2.1 applies to these schemes. For example, Ref. [43] reports $\mathcal{O}(\sqrt{N})$ parallelism, aligning with our analysis. Some studies, such as Refs. [43, 46], propose mapping, routing, and scheduling algorithms to support parallel computation. As discussed later in Section 6.2.5, our compilation scheme can be combined with such algorithms. Ref. [37] proposes a compilation scheme with a layout consisting of a large memory block, a processing unit with a limited size, and distillation factories. By leveraging memory access locality, this scheme reduces spatial overhead with only a small additional temporal cost. This is best suited for local and sequential computation, which contrasts with our target applications.

Some studies use alternative layouts. For example, Ref. [104] uses a layout that does not separate magic state factories from the computation area. This study is specific to Shor’s factoring algorithm, as is the layout. Ref. [113] proposes placing magic state factories inside the computation area statically with a motivation similar to ours. An important difference is that our proposal performs magic state distillation using the routing qubits temporarily, rather than relying on statically placed factories. Ref. [36] proposes a compilation scheme that performs computation with Clifford gates

and arbitrary-angle rotations. The arbitrary-angle rotations use a state injection protocol performed in the computation area, since it does not rely on magic state distillation.

6.2.3 Magic state distillation protocol

In the following discussions, we assume the existence of *in-place* distillation, motivated by recent progress in magic state distillation [29, 38]. In this subsection, we discuss the validity of this assumption.

Magic state cultivation [38] (Section 2.4.4) is a magic state distillation protocol. The distillation process consists of three stages: injection, cultivation, and escape. In the injection stage, a magic state is encoded into a color code non-fault-tolerantly. In the cultivation stage, the magic state is distilled by performing several rounds of Hadamard tests. At this stage, the magic state is encoded in a color code. In the escape stage, the magic state is encoded into a grafted code with a higher distance (d_g) so that we can maintain the fidelity of the magic state with error correction. The support of the grafted code is mostly square-shaped, which is attractive for in-place distillation.

Ref. [38] reports a logical error probability of 2×10^{-9} for the resultant magic state, with a physical error probability of 10^{-3} and $d_g = 15$. Moreover, in many cases, the error probability of magic states can be higher than that of data and routing qubits. For example, random circuit sampling (Section 2.5.1) performs CZ , H , S , and T gates. This suggests that magic states are used in approximately 1/6 of the gates. If there is one routing qubit for each data qubit, the error probability of a magic state can be 12 times higher than that of a data qubit or a routing qubit. This difference can be leveraged to reduce d_g and/or increase the distillation success probability. This rough estimation suggests that magic state cultivation offers in-place distillation within a certain fidelity range. Although this range is limited, future improvements in distillation protocols will expand the applicability of in-place distillation.

In this chapter, we assume that distillation is performed within a single patch to simplify the discussion. However, locality-aware Pauli-based computation is compatible with a distillation protocol that uses a few patches, though this would require increased complexity in mapping, routing, and scheduling algorithms. Whether such a setting improves applicability by reducing the error probability and increasing the distillation success probability remains an open question.

6.2.4 Quantum ISA and its implementation

The quantum ISA (Section 2.3.1) for locality-aware Pauli-based computation consists of the following operations:

1. Single-qubit $|0\rangle$ initialization,
2. Single-qubit Pauli measurement,
3. Multi-qubit Pauli measurement,
4. Two-qubit $\frac{\pi}{4}$ Pauli rotation, and
5. Single-qubit $\frac{\pi}{8}$ Pauli rotation.

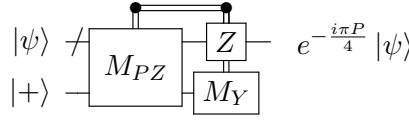


Figure 6.5: $\frac{\pi}{4}$ rotation implementation. P is a Pauli operator.

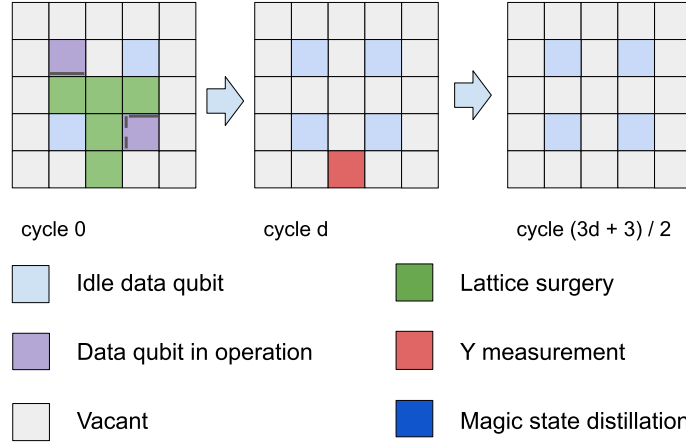


Figure 6.6: $\frac{\pi}{4}$ ZY rotation. Each square is a surface code patch (Section 2.2.4, Figure 2.8).

The first three operations are the same as those in SPC (Section 2.3.2), and we use the same implementations except for the single-qubit Y measurement, which we replace with an in-place Y measurement [114] that takes $\frac{d+3}{2}$ cycles. A two-qubit $\frac{\pi}{4}$ Pauli rotation is implemented using a multi-qubit Pauli measurement, as shown in Figure 6.5. This operation takes $\frac{3d+3}{2}$ cycles. Figure 6.6 depicts the computation process for a $\frac{\pi}{4}$ ZY rotation. A single-qubit $\frac{\pi}{8}$ rotation is implemented using a multi-qubit Pauli measurement and a magic state, as shown in Figure 6.7. This operation requires $m + \frac{3d+3}{2}$ cycles, where m is the time required for magic state distillation. Figure 6.8 depicts the computation process for a $\frac{\pi}{8}$ Z rotation. In the figure, four magic state distillation processes are performed to mitigate the low success probability of distillation. If at least one of the four distillation processes succeeds, the $\frac{\pi}{8}$ rotation completes on time. Otherwise, the system retries distillation, leading to a *runtime delay*.

When performing a multi-qubit Pauli measurement used in $\frac{\pi}{4}$ and $\frac{\pi}{8}$ Pauli rotations, we need to choose the correct edges of the target data qubit patches based on the rotation axis, as discussed in Section 2.3.2. In all examples in this subsection, a horizontal (vertical) edge of a logical data qubit patch corresponds to its logical Z (X) operator. Hence, for instance, to perform a $\frac{\pi}{4}$ XX rotation, each target data qubit patch must share one of its vertical edges with the lattice surgery ancilla patches. This ensures that each $\frac{\pi}{4}$ rotation requires a fixed number of cycles, independent of the rotation axis. The same applies to $\frac{\pi}{8}$ rotations, except in cases of magic state distillation failure. Note that magic states can be prepared with arbitrary orientation, which relaxes routing constraints for $\frac{\pi}{8}$ rotations.

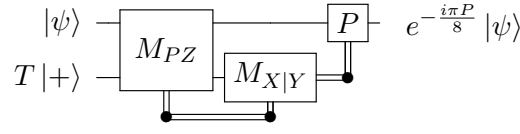


Figure 6.7: Single-qubit $\frac{\pi}{8}$ rotation implementation. P is a Pauli operator. The measurement axis of $M_{X|Y}$ depends on the measurement outcome of M_{PZ} .

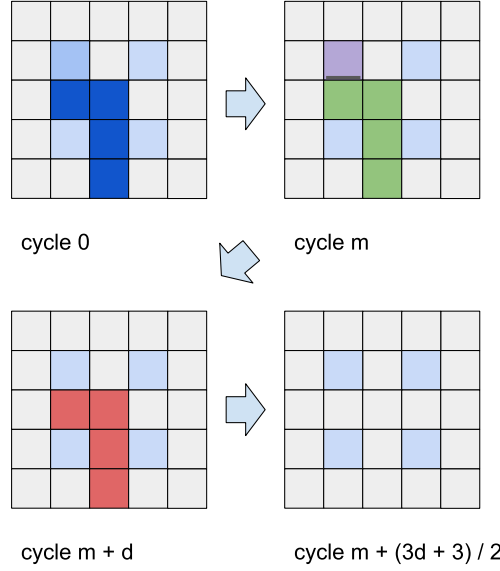


Figure 6.8: $\frac{\pi}{8}$ Z rotation. See Figure 6.6 for the coloring convention.

$$\begin{aligned}
 \text{---} \text{---} P \text{---} P' \text{---} &= \begin{cases} \text{---} P' \text{---} P \text{---} & (PP' = P'P) \\ \text{---} iPP' \text{---} P \text{---} & (PP' = -P'P) \end{cases} & \text{---} P \text{---} P' \text{---} &= \begin{cases} \text{---} P' \text{---} & (PP' = P'P) \\ \text{---} iPP' \text{---} & (PP' = -P'P) \end{cases} \\
 \text{---} P \text{---} P' \text{---} &= \begin{cases} \text{---} P' \text{---} P \text{---} & (PP' = P'P) \\ \text{---} iPP' \text{---} P \text{---} & (PP' = -P'P) \end{cases}
 \end{aligned}$$

Figure 6.9: Gate translation rules for locality-aware Pauli-based computation. P is a single-qubit Pauli operator and P' is a Pauli operator. See Figure 2.19 for the coloring convention.

6.2.5 Transpilation to the quantum ISA

A quantum program represented in the quantum IR is transpiled into the quantum ISA through the following steps. First, each arbitrary-angle Z rotation is decomposed into a sequence of Clifford and T gates using gate synthesis (Section 2.3.1). Next, each Clifford gate is decomposed into a sequence of $\frac{\pi}{4}$ rotations. Finally, single-qubit $\frac{\pi}{4}$ rotations are moved to the end of the circuit, following the rules illustrated in Figure 6.9.

We now discuss mapping, routing, and scheduling for this compilation scheme. Locality-aware Pauli-based computation is compatible with any qubit layout that supports the quantum ISA

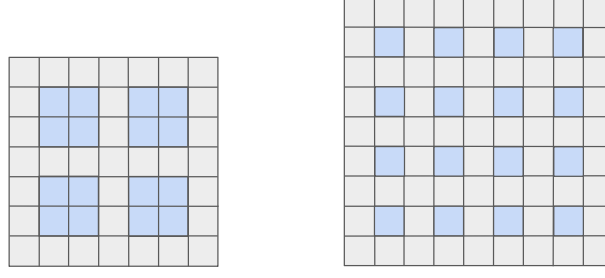


Figure 6.10: The standard layout (left) and sparse layout (right), each containing 16 logical data qubits.

implementation, including the examples shown in [Figure 6.6](#) and [Figure 6.8](#). In this chapter, we consider two layouts, *standard* and *sparse* ([Figure 6.10](#)). The standard and sparse layouts require approximately $2.25N$ and $4N$ patches, respectively, where N is the number of data qubits. Unlike SPC, efficient mapping, routing, and scheduling are crucial for locality-aware Pauli-based computation, as it aims to execute many gates simultaneously by leveraging locality. In [Section 6.3](#), we implement basic mapping, routing, and scheduling strategies. We expect that locality-aware Pauli-based computation will perform better with more sophisticated algorithms such as those proposed in Refs. [\[43, 46\]](#), but we leave a detailed investigation for future work.

6.3 Performance evaluation

This section presents a performance evaluation of locality-aware Pauli-based computation. We begin by describing the design and implementation of the scheduler and runtime simulator in [Section 6.3.1](#) and [Section 6.3.2](#). We then describe the simulation parameters in [Section 6.3.3](#). Finally, we present the simulation results in [Section 6.3.4](#). The scheduler and runtime simulator are publicly available [\[115\]](#).

6.3.1 Scheduler implementation

[Figure 6.11](#) shows the system configuration used in the simulation. The scheduler outputs the scheduling result, given a QASM program and qubit mapping file. It implements gate synthesis, mapping, routing, and scheduling algorithms, described below.

Gate synthesis: For the simulation, we emulate the mixed diagonal algorithm by replacing an arbitrary-angle Z rotation $e^{i\theta Z}$ with a sequence of l randomly chosen $\frac{\pi}{8}$ rotations, followed by two randomly chosen $\frac{\pi}{4}$ rotations. The value of l is sampled from a normal distribution with mean 1.5δ , where $2^{-\delta}$ is the target precision of the rotation angle.

Mapping: As shown in [Figure 6.11](#), the mapping configuration is provided externally to the scheduler to simplify its implementation.

Routing: To execute a two-qubit $\frac{\pi}{4}$ rotation, we find a path between the two patches using a breadth-first search algorithm. Ancilla patches near the data qubits may be required depending on the rotation axis. An example is shown in [Figure 6.6](#). Additionally, one ancilla patch is required for a Y measurement.

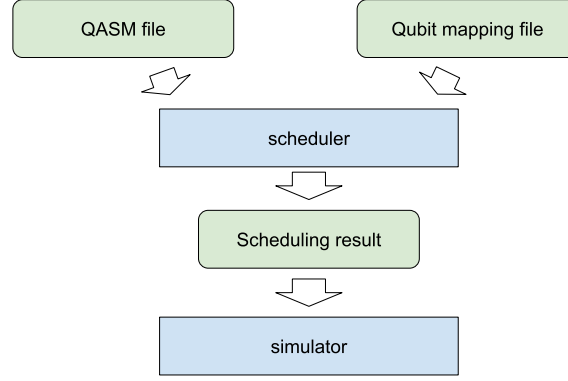


Figure 6.11: System configuration of the scheduler and simulator used in the performance evaluation.

Algorithm 7 Scheduling algorithm.

```

1: for  $i$  in instructions do
2:    $support = \text{support qubits of } i$ 
3:    $cycle = \text{maximal scheduled cycles on } support$ 
4:   while true do
5:     if routing for  $i$  is possible at  $cycle$  then
6:       Commit the schedule for  $i$  starting at  $cycle$ .
7:       break
8:     end if
9:      $cycle += 1$ 
10:  end while
11: end for
  
```

For a single-qubit $\frac{\pi}{8}$ rotation, we allocate D connected patches for magic state distillation, where D is a simulator parameter. The allocation is computed using a breadth-first search algorithm to minimize the diameter of the distillation area. Similar to the two-qubit $\frac{\pi}{4}$ rotation case, ancilla patches near the data qubits may be required depending on the rotation axis.

Scheduling: We use a greedy algorithm (Algorithm 7) to schedule operations. The algorithm takes as input a sequence of quantum ISA instructions, denoted *instructions*, ordered by gate dependencies. It produces a schedule consisting of *microinstructions* assigned to each patch and cycle. Each microinstruction falls into one of the following categories: *idle data qubit*, *data qubit in operation*, *vacant*, *lattice surgery*, *Y measurement*, or *magic state distillation*, as indicated by the color coding in Figure 6.6. Figure 6.12 shows a snapshot of the schedule for 6×6 random circuit sampling at a specific cycle.

6.3.2 Simulator implementation

The runtime simulator estimates the execution time based on a given schedule as depicted in Figure 6.11. Most aspects of the simulation are straightforward; the only nontrivial component is the calculation of the runtime delay. Magic state distillation is the only source of runtime delay. In this simulation, we model the time cost of magic state distillation as $m \cdot G(p)$, where m is the

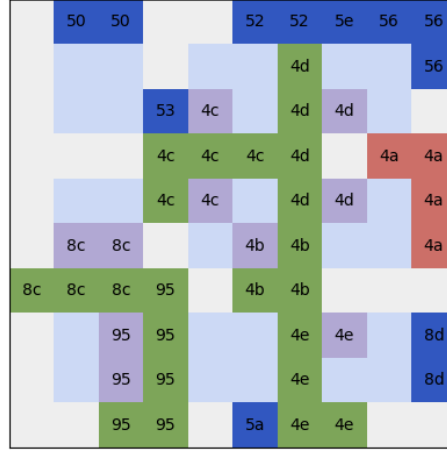


Figure 6.12: Schedule for 6×6 random circuit sampling. Each number represents an instruction ID. See Figure 6.6 for the color coding scheme.

time required for a single distillation round and $G(p)$ is a geometric random variable with success probability p , sampled independently for each event.

As discussed in Section 6.2.1, runtime delay can propagate. The simulation uses the following runtime delay propagation rules.

- Any microinstruction on a patch except for *idle data qubit* and *vacant* must wait for all preceding microinstructions on the same patch to complete.
- *Lattice surgery* and *data qubit in operation* microinstructions must wait for all preceding microinstructions on the involved patches to complete.

6.3.3 Parameter settings

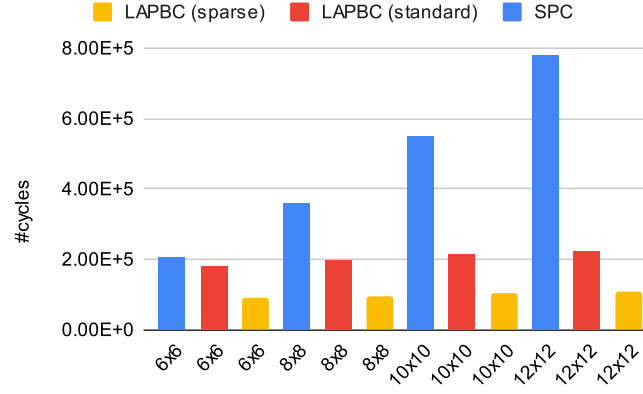
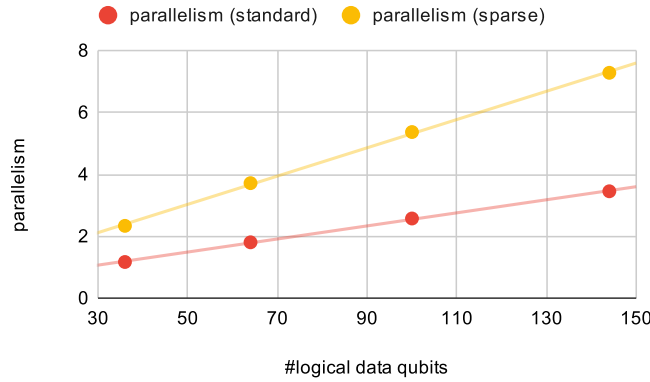
Table 6.1 summarizes the parameters used in the following simulations. We use the same code distance, $d = 15$, across all simulations. In general, larger problem sizes require larger code distances than smaller ones. In this chapter, however, we use the same code distance for the following reasons. First, the differences in problem size are relatively small in our case. With a physical error probability 10^{-3} and a surface code threshold 10^{-2} , increasing code distance by 2 corresponds to increasing the problem size by a factor of 10 in qubit-cycles. In comparison, the 12×12 instance is only four times larger than the 6×6 instance. Second, comparing results with the same code distance is significantly easier than comparing those with different code distances. We apply the same reasoning to the use of common values for m , p_{success} , and ρ across all simulations. Note also that the code distance $d = 15$ provides sufficient fidelity for all simulations presented below.

6.3.4 Results

We performed simulations of random circuit sampling (Section 2.5.1) and the Hamiltonian simulation with Trotterization (Section 2.5.3). These applications have natural 2D structures, and we performed manual qubit mapping based on these structures. Figure 6.13 shows the simulation results for random

Table 6.1: Parameter settings used in the simulations.

symbol	description	value
d	code distance	15
m	distillation time cost	27
p_{success}	distillation success probability	0.25
ρ	arbitrary-angle rotation precision	10^{-7}

**Figure 6.13:** Simulation results for random circuit sampling with 500 layers.**Figure 6.14:** Parallelism of locality-aware Pauli-based computation with random circuit sampling.

circuit sampling with 500 layers. While the cycle count for SPC grows approximately proportionally with the number of logical data qubits, that for locality-aware Pauli-based computation—using both standard and sparse layouts—increases more gradually due to greater gate-level parallelism. Note that the results for locality-aware Pauli-based computation account for runtime delays from magic state distillation, whereas the results for SPC do not. This is because runtime delays are easier to mitigate for SPC due to its sequential nature and the small spatial cost of magic state distillation. Figure 6.14 shows the parallelism of locality-aware Pauli-based computation, where parallelism is defined as the number of cycles required for SPC divided by the number of cycles required for locality-aware Pauli-based computation. Parallelism scales linearly with the number of data qubits N , following a trend of the form $aN + b$ for some constants a and b , and surpasses the $\mathcal{O}(\sqrt{N})$ scaling of the conventional layout (Section 6.2.1). Figure 6.15 shows the Hamiltonian simulation results, which exhibit a similar trend.

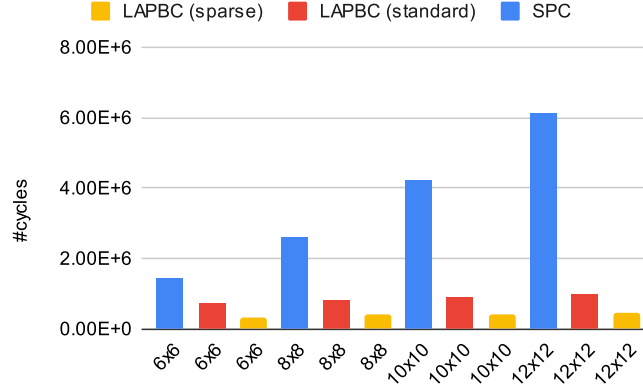


Figure 6.15: Simulation results for Hamiltonian simulation with Trotterization.

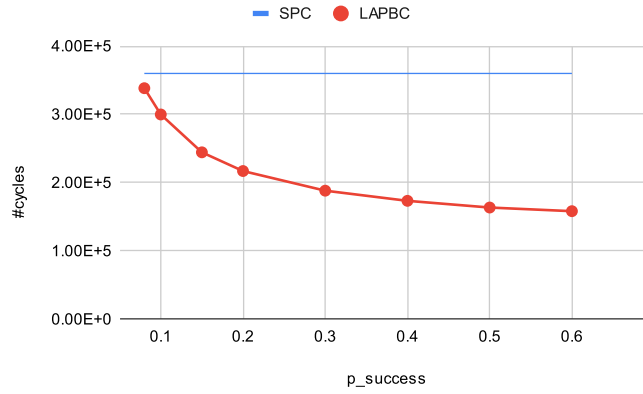


Figure 6.16: Simulation results for 8×8 random circuit sampling with varying p_{success} .

For 6×6 (12×12) random circuit sampling, locality-aware Pauli-based computation with the standard layout achieves a 14% (71%) reduction in execution time relative to SPC. Similarly, for 6×6 (12×12) Hamiltonian simulation with Trotterization, it achieves a 48% (84%) reduction. The spatial overhead of the standard layout is comparable to that of SPC, so these improvements are obtained with little or no additional spatial cost. For 6×6 (12×12) random circuit sampling, the sparse layout achieves a 57% (86%) reduction in execution time relative to SPC. For Hamiltonian simulation, the reductions are 76% and 93%, respectively. The sparse layout incurs approximately twice the spatial overhead of SPC.

Figure 6.16 shows the simulation results for 8×8 random circuit sampling using the standard layout, with varying values of p_{success} . The results demonstrate that the cycle count is sensitive to p_{success} , indicating that the distillation cost remains a significant factor for highly parallel algorithms. Moreover, in this case, increasing p_{success} up to 0.4 significantly reduces the overall time cost, whereas further increases have a negligible impact.

6.4 Conclusion

In this chapter, we proposed locality-aware Pauli-based computation, a novel compilation scheme optimized for highly parallel quantum algorithms. By preserving gate locality in the original

algorithm and performing magic state distillation locally, the scheme enables the parallel execution of many T gates.

Numerical simulations in [Section 6.3](#) show that, for 6×6 (12×12) random circuit sampling, locality-aware Pauli-based computation achieves a 14% (71%) reduction in execution time relative to SPC, with little or no additional spatial overhead. Similarly, for 6×6 (12×12) Hamiltonian simulation with Trotterization, it achieves a 48% (84%) reduction. Even greater reductions can be achieved with increased spatial overhead, as demonstrated by the sparse layout. These results demonstrate that locality-aware Pauli-based computation scales favorably in execution time as the number of qubits increases. Therefore, we conclude that locality-aware Pauli-based computation will become a standard compilation scheme targeting highly parallel quantum algorithms when high-performance magic state distillation is available.

Our scheduler and runtime simulator reveal the relationship between the magic state distillation cost—including the distillation success probability—and the overall computational cost for a given quantum algorithm. This provides useful design guidelines for developing magic state distillation protocols.

When magic state distillation was hundreds of times more expensive than Clifford gates, it was reasonable to treat the problems of magic state preparation and consumption separately. However, as the cost of distillation continues to decrease, considering preparation and consumption jointly becomes increasingly important. This chapter highlights the effectiveness of such an integrated approach in the design of compilation schemes.

The mapping and routing algorithms implemented for the numerical simulations are simple. We expect that locality-aware Pauli-based computation will achieve better performance with more sophisticated algorithms such as those proposed in Refs. [\[43, 46\]](#), but we leave a detailed investigation to future work. Integrating the factory-less layout with the direct Clifford+ T compilation [\[116\]](#) presents another interesting direction.

In this chapter, we assume that each distillation is performed within a single patch. However, locality-aware Pauli-based computation is also compatible with distillation protocols that use multiple patches, though this would introduce additional complexity. Since routing qubits are temporarily used for distillation, the number of patches required by a distillation factory is a more important factor than the total number of physical qubits it occupies. Whether such a setting is more effective in practice—by reducing the logical error probability or increasing the distillation success probability—remains an important open question.

Chapter 7

Conclusion

In this thesis, we have reduced FTQC overhead by focusing on the preparation and consumption of magic states. We first focused on magic state preparation and proposed efficient distillation protocols that use physical-level distillation. In [chapter 3](#), we proposed *(0+1)-level distillation*, a two-level distillation protocol, to prepare high-fidelity magic states using zero-level distillation. In [chapter 4](#), we proposed *magic state cultivation with lattice surgery*, which lowers the distillation overhead of magic state cultivation by using lattice surgery and the rotated surface code. These protocols greatly reduce the distillation overhead, making it possible to prepare many magic states simultaneously with moderate spacetime cost.

We then turned to FTQC architectures designed to efficiently consume the magic states prepared by these distillation protocols in parallel. In [chapter 5](#), we introduced a pool of magic states that mitigates the negative performance impact of magic state distillation failures. This becomes especially important with physical-level distillation protocols, because they enable greater parallelism while also exhibiting higher failure probabilities than logical-level distillation protocols. Numerical simulations show that the negative impact is amplified by parallel execution, even when the failure probability is small, and that the pool successfully reduces the total FTQC overhead.

In [chapter 6](#), we introduced *locality-aware Pauli-based computation* (LAPBC), a compilation scheme optimized for highly parallel quantum algorithms. By preserving gate locality in the original algorithm and performing magic state distillation locally, the scheme enables the parallel execution of many T gates. Numerical simulations show that the scheme shortens the execution time compared to SPC [26] with little or no additional spatial overhead. Simulation results further indicate that the scheme scales favorably with system size.

In 2012, Jones *et al.* [19] reported that 90% of physical qubits were devoted to distillation in their resource estimation of 1024-bit integer factoring. While many distillation protocols had been proposed, distillation remained a major bottleneck in fault-tolerant quantum computation. This observation motivated architectural designs optimized under the constraint of expensive magic state distillation. For example, SPC focuses on efficiently executing logical gates sequentially, while the STAR architecture [34] offers favorable spacetime overhead by avoiding T magic state distillation.

Physical-level distillation substantially lowers the overhead associated with magic state distillation. This has been established by prior work, including zero-level distillation [29] and magic

state cultivation [38], and is further developed in this thesis through our contributions to magic state preparation. This allows FTQC architecture designers to use low-cost magic states. Gidney [117] reported a 95% qubit count reduction for 2048-bit integer factoring compared to earlier estimates [104]. Magic state cultivation is one of the key drivers of the reduction; magic state distillation factories occupy less than 20% of physical qubits, whereas data qubits are encoded in the yoked surface code [55], a compressed form of the surface code. In this thesis, we translate this distillation efficiency gain into a reduced total FTQC overhead by consuming many magic states in parallel. As a result, our contributions reduce the overall computation overhead and advance the practical realization of resource-efficient fault-tolerant quantum computation.

This thesis clarifies that the traditional view of magic state distillation as the dominant source of FTQC overhead no longer holds in the near-term FTQC regimes, including megaquop- and gigaquop-scale computations [48]. When distillation overhead is reduced, architectural and compilation decisions that were previously considered optimal may become suboptimal, and new trade-offs emerge. In such regimes, efficient consumption of magic states becomes as important as their preparation, as concretely demonstrated by the impact of distillation failures and magic state transfer costs in highly parallel quantum algorithms. These results indicate that FTQC architectures and compilation schemes should be evaluated from a system-wide perspective, in which magic state preparation and consumption are considered jointly rather than as independent components.

In this thesis, we have primarily focused on highly parallel quantum algorithms, as they are practically important and place strong demands on the efficient preparation and consumption of magic states. We believe, however, that the implications of this thesis extend to other settings as well. Recently, FTQC architectures based on QEC codes that are more space-efficient than the surface code have been actively investigated [55, 118–121], while some of these codes require additional hardware capabilities compared to those assumed in this thesis. These QEC codes substantially reduce the spatial overhead required to store data qubits. As a consequence, the relative importance of magic state preparation and consumption increases, even in regimes where parallelism is limited. Gidney’s resource estimation for integer factoring, which uses both the yoked surface code and magic state cultivation, provides a concrete example. Similarly, transversal architectures on neutral-atom hardware platforms [122] enable significantly faster logical Clifford gate execution than lattice surgery, further shifting the performance bottleneck toward non-Clifford resources. Extending this system-level perspective to emerging FTQC architectures would be an interesting direction for future investigation.

List of Achievements

Publications

Publications included in this thesis

1. Y. Hirano, T. Itogawa, and K. Fujii, “Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States,” 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 843–853. IEEE (2024).
2. Y. Hirano, R. Toshio, T. Itogawa, and K. Fujii, “Efficient magic state cultivation with lattice surgery,” *arXiv*: 2510.24615 (2025).
3. Y. Hirano, Y. Suzuki, and K. Fujii, “MagicPool: Dealing with Magic State Distillation Failures on Large-Scale Fault-Tolerant Quantum Computer,” *arXiv*: 2407.07394 (2024).
4. Y. Hirano and K. Fujii, “Locality-Aware Pauli-Based Computation for Local Magic State Preparation,” 2025 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 670–680. IEEE (2025).

Publications not included in this thesis

1. T. Itogawa, Y. Takada, Y. Hirano, and K. Fujii, “Efficient Magic State Distillation by Zero-Level Distillation,” *PRX Quantum* **6**, 020356 (2025).
2. Y. Ibe, Y. Hirano, Y. Ozu, T. Kawakubo, and K. Fujii, “Measurement-Based Fault-Tolerant Quantum Computation on High-Connectivity Devices: A Resource-Efficient Approach toward Early FTQC,” *arXiv*: 2510.18652 (2025).
3. S. Sunami, Y. Hirano, T. Hinokuma, and H. Yamasaki, “Entanglement boosting: Low-volume logical Bell pair preparation for distributed fault-tolerant quantum computation,” *arXiv*: 2511.10729 (2025).

Conference presentations

Oral presentations

1. Y. Hirano, T. Itogawa, and K. Fujii, “Leveraging Zero-Level Distillation to Generate High-Fidelity Magic States,” 2024 IEEE International Conference on Quantum Computing and

Engineering (QCE), Montreal, Canada, Sep 15–20, 2024.

2. Y. Hirano and K. Fujii, “Locality-Aware Pauli-Based Computation for Local Magic State Preparation,” Workshop on Fault-Tolerant Quantum Computer System Architecture, Tokyo, Japan, Jun 22, 2025.
3. Y. Hirano and K. Fujii, “Locality-Aware Pauli-Based Computation for Local Magic State Preparation,” 2025 IEEE International Conference on Quantum Computing and Engineering (QCE), Albuquerque, New Mexico, USA, Aug 31–Sep 5, 2025.

Poster presentations

1. Y. Hirano and K. Fujii, “Locality-Aware Pauli-Based Computation for Local Magic State Preparation,” The International Symposium on Quantum Science, Technology and Innovation, Osaka, Japan, Jul 29–Aug 2, 2025 (Poster Award).
2. Y. Hirano and K. Fujii, “Locality-Aware Pauli-Based Computation for Local Magic State Preparation,” International Conference on Quantum Error Correction, New Haven, Connecticut, USA, Aug 11–15, 2025.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Prof. Keisuke Fujii, my PhD advisor, for his continuous support throughout my doctoral studies. I began studying quantum computing at the start of my PhD (and shortly before), and his guidance, insight, and encouragement were essential in helping me build a strong foundation in the field and complete this thesis.

I am also grateful to my collaborators and lab members for their valuable contributions and many fruitful technical discussions. Their expertise and feedback greatly influenced the direction and quality of this work.

I would like to thank Dr. Takefumi Miyoshi, who introduced me to the field of quantum computing and motivated me to pursue research in this area.

Last but not least, I would like to thank my wife, who has always supported me in pursuing my PhD. Her understanding and encouragement made it possible for me to continue my studies.

Bibliography

- [1] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.
- [2] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.
- [3] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.
- [4] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.
- [5] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.
- [6] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.
- [7] Aram W Harrow, Avinandan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.
- [8] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David Buell, Brian Burkett, Yu Chen, Jimmy Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Michael Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew Harrigan, Michael Hartmann, Alan Ho, Markus Rudolf Hoffmann, Trent Huang, Travis Humble, Sergei Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, Dave Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod Ryan McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin Jeffery Sung, Matt Trevithick, Amit Vainsencher, Benjamin Villalonga, Ted White, Z. Jamie Yao, Ping

- Yeh, Adam Zalcman, Hartmut Neven, and John Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 2019.
- [9] Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th conference on foundations of computer science*, pages 56–65. IEEE, 1996.
- [10] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.
- [11] Google Quantum AI and Collaborators. Quantum error correction below the surface code threshold. *Nature*, 638(8052):920–926, 2025.
- [12] Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.
- [13] Daniel Gottesman. The heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*, 1998.
- [14] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, Feb 2005.
- [15] Jonas T. Anderson, Guillaume Duclos-Cianci, and David Poulin. Fault-tolerant conversion between the steane and reed-muller quantum codes. *Phys. Rev. Lett.*, 113:080501, Aug 2014.
- [16] Héctor Bombín. Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes. *New Journal of Physics*, 17(8):083002, aug 2015.
- [17] Héctor Bombín. Dimensional jump in quantum error correction. *New Journal of Physics*, 18(4):043038, apr 2016.
- [18] Michael E. Beverland, Aleksander Kubica, and Krysta M. Svore. Cost of universality: A comparative study of the overhead of state distillation and code switching with color codes. *PRX Quantum*, 2:020341, Jun 2021.
- [19] N Cody Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical Review X*, 2(3):031007, 2012.
- [20] Emanuel Knill, Raymond Laflamme, and Wojciech H Zurek. Resilient quantum computation. *Science*, 279(5349):342–345, 1998.
- [21] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(5):052329, 2012.
- [22] Adam M Meier, Bryan Eastin, and Emanuel Knill. Magic-state distillation with the four-qubit code. *arXiv preprint arXiv:1204.4221*, 2012.

- [23] Hayato Goto. Minimizing resource overheads for fault-tolerant preparation of encoded states of the steane code. *Scientific Reports volume 6, Article number: 19578*, 2016.
- [24] Earl T. Campbell and Mark Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Phys. Rev. A*, 95:022316, Feb 2017.
- [25] Craig Gidney and Austin G. Fowler. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation. *Quantum*, 3:135, April 2019.
- [26] Daniel Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, March 2019.
- [27] Daniel Litinski. Magic State Distillation: Not as Costly as You Think. *Quantum*, 3:205, December 2019.
- [28] Christopher Chamberland and Kyungjoo Noh. Very low overhead fault-tolerant magic state preparation using redundant ancilla encoding and flag qubits. *npj Quantum Information*, 6(1):91, 2020.
- [29] Tomohiro Itogawa, Yugo Takada, Yutaka Hirano, and Keisuke Fujii. Efficient magic state distillation by zero-level distillation. *PRX Quantum*, 6(2):020356, 2025.
- [30] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, January 2003.
- [31] Sergey Bravyi and Robert Raussendorf. Measurement-based quantum computation with the toric code states. *Physical Review A—Atomic, Molecular, and Optical Physics*, 76(2):022304, 2007.
- [32] Matthew B Hastings and Jeongwan Haah. Distillation with sublogarithmic overhead. *Physical review letters*, 120(5):050504, 2018.
- [33] Michael E. Beverland, Prakash Murali, Matthias Troyer, Krysta M. Svore, Torsten Hoefler, Vadym Kliuchnikov, Guang Hao Low, Mathias Soeken, Aarthi Sundaram, and Alexander Vaschillo. Assessing requirements to scale to practical quantum advantage, 2022.
- [34] Yutaro Akahoshi, Kazunori Maruyama, Hirotaka Oshima, Shintaro Sato, and Keisuke Fujii. Partially fault-tolerant quantum computing architecture with error-corrected clifford gates and space-time efficient analog rotations. *PRX quantum*, 5(1):010337, 2024.
- [35] Riki Toshio, Yutaro Akahoshi, Jun Fujisaki, Hirotaka Oshima, Shintaro Sato, and Keisuke Fujii. Practical quantum advantage on partially fault-tolerant quantum computer. *Physical Review X*, 15(2):021057, 2025.
- [36] Yutaro Akahoshi, Riki Toshio, Jun Fujisaki, Hirotaka Oshima, Shintaro Sato, and Keisuke Fujii. Compilation of trotter-based time evolution for partially fault-tolerant quantum computing architecture. *PRX Quantum*, 6:040319, Oct 2025.

- [37] Takumi Kobori, Yasunari Suzuki, Yosuke Ueno, Teruo Tanimoto, Syngé Todo, and Yuuki Tokunaga. Lsqca: Resource-efficient load/store architecture for limited-scale fault-tolerant quantum computing. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 304–320. IEEE, 2025.
- [38] Craig Gidney, Noah Shetty, and Cody Jones. Magic state cultivation: growing T states as cheap as CNOT gates, 2024.
- [39] Hector Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical review letters*, 97(18):180501, 2006.
- [40] Héctor Bombín and Miguel A Martin-Delgado. Optimal resources for topological two-dimensional stabilizer codes: Comparative study. *Physical Review A—Atomic, Molecular, and Optical Physics*, 76(1):012305, 2007.
- [41] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, dec 2012.
- [42] Hendrik Poulsen Nautrup, Nicolai Friis, and Hans J Briegel. Fault-tolerant interface between quantum memories and quantum processors. *Nature communications*, 8(1):1321, 2017.
- [43] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. Surface code compilation via edge-disjoint paths. *PRX Quantum*, 3:020342, May 2022.
- [44] Christopher Chamberland and Earl T. Campbell. Universal quantum computing with twist-free and temporally encoded lattice surgery. *PRX Quantum*, 3:010331, Feb 2022.
- [45] Tyler LeBlond, Christopher Dean, George Watkins, and Ryan Bennink. Realistic cost to execute practical quantum circuits using direct clifford+ t lattice surgery compilation. *ACM Transactions on Quantum Computing*, 5(4):1–28, 2024.
- [46] Abtin Molavi, Amanda Xu, Swamit Tannu, and Aws Albarghouthi. Dependency-aware compilation for surface code quantum architectures, 2024.
- [47] Kou Hamada, Yasunari Suzuki, and Yuuki Tokunaga. Efficient and high-performance routing of lattice-surgery paths on three-dimensional lattice, 2024.
- [48] John Preskill. Beyond NISQ: The megauoq machine. *ACM Transactions on Quantum Computing*, 6(3):1–7, 2025.
- [49] Yutaka Hirano, Tomohiro Itogawa, and Keisuke Fujii. Leveraging zero-level distillation to generate high-fidelity magic states. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 843–853. IEEE, 2024.
- [50] Yutaka Hirano, Riki Toshio, Tomohiro Itogawa, and Keisuke Fujii. Efficient magic state cultivation with lattice surgery. *arXiv preprint arXiv:2510.24615*, 2025.

- [51] Yutaka Hirano, Yasunari Suzuki, and Keisuke Fujii. Magicpool: Dealing with magic state distillation failures on large-scale fault-tolerant quantum computer, 2024.
- [52] Yutaka Hirano and Keisuke Fujii. Locality-aware pauli-based computation for local magic state preparation. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pages 670–680, 2025.
- [53] P Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing. *arXiv preprint quant-ph/9906054*, 1999.
- [54] Daniel Gottesman. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.
- [55] Craig Gidney, Michael Newman, Peter Brooks, and Cody Jones. Yoked surface codes. *Nature Communications*, 16(1):4498, 2025.
- [56] A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996.
- [57] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996.
- [58] Andrew Steane. Multiple-particle interference and quantum error correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 452(1954):2551–2577, 1996.
- [59] Leon Riesebos, Xiang Fu, Savvas Varsamopoulos, Carmen G Almudever, and Koen Bertels. Pauli frames for quantum computer architectures. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.
- [60] Matt McEwen, Dave Bacon, and Craig Gidney. Relaxing Hardware Requirements for Surface Code Circuits using Time-dynamics. *Quantum*, 7:1172, November 2023.
- [61] Peter-Jan HS Derks, Alex Townsend-Teague, Ansgar G Burchards, and Jens Eisert. Designing fault-tolerant circuits using detector error models. *arXiv preprint arXiv:2407.13826*, 2024.
- [62] R Tanner. A recursive approach to low complexity codes. *IEEE Transactions on information theory*, 27(5):533–547, 2003.
- [63] H-A Loeliger. An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1):28–41, 2004.
- [64] Benjamin J Brown. Conservation laws and quantum error correction: Toward a generalized matching decoder. *IEEE BITS the Information Theory Magazine*, 2(3):5–19, 2023.
- [65] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error rate, 1999.

- [66] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997.
- [67] Panos Aliferis, Daniel Gottesman, and John Preskill. Quantum accuracy threshold for concatenated distance-3 codes. *arXiv preprint quant-ph/0504218*, 2005.
- [68] Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Phys. Rev. Lett.*, 102:110502, Mar 2009.
- [69] S. B. Bravyi and A. Yu. Kitaev. Quantum codes on a lattice with boundary, 1998.
- [70] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012.
- [71] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.
- [72] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.
- [73] Austin G Fowler. Optimal complexity correction of correlated errors in the surface code. *arXiv preprint arXiv:1310.0863*, 2013.
- [74] Daniel Litinski and Felix von Oppen. Lattice surgery with a twist: Simplifying clifford gates of surface codes. *Quantum*, 2:62, 2018.
- [75] Adrian Hutter, James R Wootton, and Daniel Loss. Efficient markov chain monte carlo algorithm for the surface code. *Physical Review A*, 89(2):022326, 2014.
- [76] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, RWDSL2018, New York, NY, USA, 2018. Association for Computing Machinery.
- [77] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. *SIGPLAN Not.*, 48(6):333–342, June 2013.
- [78] QIR Alliance. *QIR Specification*, 2021. Also see <https://qir-alliance.org>.
- [79] Vadym Kliuchnikov, Kristin Lauter, Romy Minko, Adam Paetznick, and Christophe Petit. Shorter quantum circuits via single-qubit gate approximation. *Quantum*, 7:1208, December 2023.
- [80] Jeongwan Haah and Matthew B Hastings. Codes and protocols for distilling t , controlled- s , and toffoli gates. *Quantum*, 2:71, 2018.

- [81] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Phys. Rev. X*, 8:041015, Oct 2018.
- [82] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995.
- [83] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2:040203, Dec 2021.
- [84] Nobuyuki Yoshioka, Tsuyoshi Okubo, Yasunari Suzuki, Yuki Koizumi, and Wataru Mizukami. Hunting for quantum-classical crossover in condensed matter problems, 2022.
- [85] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh V. Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 15:159–163, 2018.
- [86] Alán Aspuru-Guzik, Anthony D. Dutoi, Peter J. Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.
- [87] Natalie J Pearson. *Simulating Many-Body Quantum Systems: Quantum Algorithms and Experimental Realisation*. PhD thesis, ETH Zurich, 2020.
- [88] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.
- [89] Keisuke Fujii. *Quantum Computation with Topological Codes: from qubit to topological fault-tolerance*, volume 8. Springer, 2015.
- [90] Daniel Herr, Franco Nori, and Simon J Devitt. Lattice surgery translation for quantum computation. *New Journal of Physics*, 19(1):013034, jan 2017.
- [91] Daniel Litinski. Resource-cost estimates for magic state distillation.
- [92] Yotam Vaknin, Shoham Jacoby, Arne Grimsmo, and Alex Retzker. Magic state cultivation on the surface code. *arXiv preprint arXiv:2502.01743*, 2025.
- [93] Zi-Han Chen, Ming-Cheng Chen, Chao-Yang Lu, and Jian-Wei Pan. Efficient magic state cultivation on $\mathbb{RP}^2$. *arXiv preprint arXiv:2503.18657*, 2025.
- [94] Kaavya Sahay, Pei-Kai Tsai, Kathleen Chang, Qile Su, Thomas B Smith, Shraddha Singh, and Shruti Puri. Fold-transversal surface code cultivation. *arXiv preprint arXiv:2509.05212*, 2025.
- [95] Ying Li. A magic state’s fidelity can be superior to the operations that created it. *New Journal of Physics*, 17(2):023037, 2015.
- [96] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021.
- [97] Craig Gidney and Cody Jones. New circuits and an open source decoder for the color code. *arXiv preprint arXiv:2312.08813*, 2023.

- [98] Oscar Higgott and Craig Gidney. Sparse Blossom: correcting a million errors per core second with minimum-weight matching. *Quantum*, 9:1600, January 2025.
- [99] Samuel Smith, Benjamin Brown, and Stephen Bartlett. Mitigating errors in logical qubits. *Communications Physics*, 7, 11 2024.
- [100] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013.
- [101] Peter Selinger. Quantum circuits of t -depth one. *Phys. Rev. A*, 87:042302, Apr 2013.
- [102] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time t -depth optimization of clifford+ t circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.
- [103] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, NY, USA, 1967. Association for Computing Machinery.
- [104] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021.
- [105] Mengyu Zhang, Lei Xie, Zhenxing Zhang, Qiaonian Yu, Guanglei Xi, Hualiang Zhang, Fuming Liu, Yarui Zheng, Yicong Zheng, and Shengyu Zhang. Exploiting different levels of parallelism in the quantum control microarchitecture for superconducting qubits. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 898–911, New York, NY, USA, 2021. Association for Computing Machinery.
- [106] Héctor Bombín, Mihir Pant, Sam Roberts, and Karthik I. Seetharam. Fault-tolerant postselection for low-overhead magic state preparation. *PRX Quantum*, 5:010302, Jan 2024.
- [107] Austin G. Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery, 2019.
- [108] Craig Gidney. Cleaner magic states with hook injection, 2023.
- [109] Shraddha Singh, Andrew S. Darmawan, Benjamin J. Brown, and Shruti Puri. High-fidelity magic-state preparation with a biased-noise architecture. *Phys. Rev. A*, 105:052410, May 2022.
- [110] Charles H. Bennett, Gilles Brassard, Sandu Popescu, Benjamin Schumacher, John A. Smolin, and William K. Wootters. Purification of noisy entanglement and faithful teleportation via noisy channels. *Phys. Rev. Lett.*, 76:722–725, Jan 1996.
- [111] Ying Li, Sean D Barrett, Thomas M Stace, and Simon C Benjamin. Long range failure-tolerant entanglement distribution. *New Journal of Physics*, 15(2):023012, feb 2013.

- [112] Filip Rozpędek, Thomas Schiet, Le Phuc Thinh, David Elkouss, Andrew C. Doherty, and Stephanie Wehner. Optimizing practical entanglement distillation. *Phys. Rev. A*, 97:062333, Jun 2018.
- [113] Adam Holmes, Yongshan Ding, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems*, 67:56–70, 2019.
- [114] Craig Gidney. Inplace access to the surface code y basis. *Quantum*, 8:1310, 2024.
- [115] Yutaka Hirano. Locality-aware pauli-based computation simulation source code.
- [116] Tyler LeBlond and Ryan S Bennink. Quantum resource comparison for two leading surface code lattice surgery approaches. *arXiv preprint arXiv:2506.08182*, 2025.
- [117] Craig Gidney. How to factor 2048 bit rsa integers with less than a million noisy qubits. *arXiv preprint arXiv:2505.15917*, 2025.
- [118] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024.
- [119] Joshua Vizslai, Willers Yang, Sophia Fuhui Lin, Junyu Liu, Natalia Nottingham, Jonathan M Baker, and Frederic T Chong. Matching generalized-bicycle codes to neutral atoms for low-overhead fault-tolerance. *arXiv preprint arXiv:2311.16980*, 2023.
- [120] Qian Xu, J Pablo Bonilla Ataides, Christopher A Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D Lukin, Liang Jiang, and Hengyun Zhou. Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. *Nature Physics*, 20(7):1084–1090, 2024.
- [121] Maxime A Tremblay, Nicolas Delfosse, and Michael E Beverland. Constant-overhead quantum error correction with thin planar connectivity. *Physical Review Letters*, 129(5):050504, 2022.
- [122] Hengyun Zhou, Casey Duckering, Chen Zhao, Dolev Bluvstein, Madelyn Cain, Aleksander Kubica, Sheng-Tao Wang, and Mikhail D Lukin. Resource analysis of low-overhead transversal architectures for reconfigurable atom arrays. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pages 1432–1448, 2025.