

Compilation of Trotter-Based Time Evolution for Partially Fault-Tolerant Quantum Computing Architecture

Yutaro Akahoshi^{1,2,*}, Riki Toshio^{1,2}, Jun Fujisaki^{1,2}, Hirotaka Oshima^{1,2}, Shintaro Sato^{1,2}, and Keisuke Fujii^{2,3,4,5}

¹Quantum Laboratory, Fujitsu Research, Fujitsu Limited, 4-1-1 Kawasaki, Kanagawa 211-8588, Japan

²Fujitsu Quantum Computing Joint Research Division, Center for Quantum Information and Quantum Biology, Osaka University, 1-2 Machikaneyama, Toyonaka, Osaka 565-8531, Japan

³Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

⁴Center for Quantum Information and Quantum Biology, Osaka University, 560-0043, Japan

⁵RIKEN Center for Quantum Computing (RQC), Wako Saitama 351-0198, Japan



(Received 21 October 2024; revised 12 February 2025; accepted 29 August 2025; published 28 October 2025)

Achieving practical quantum speedup with limited resources is a crucial challenge in both academic and industrial communities. To address this, a partially fault-tolerant quantum computing architecture called “space-time efficient analog rotation quantum computing architecture” (STAR architecture) been recently been proposed. This architecture focuses on minimizing resource requirements while maximizing the precision of non-Clifford gates, essential for universal quantum computation. However, nondeterministic processes such as the repeat-until-success (RUS) protocol and state injection can introduce significant computational overhead. Therefore, optimizing the logical circuit to minimize this overhead by using efficient fault-tolerant operations is essential. This paper presents an efficient method for simulating the time evolution of the two-dimensional (2D) Hubbard-model Hamiltonian, a promising application of the STAR architecture. We present two techniques, the parallel-injection protocol and adaptive injection-region updating, to reduce unnecessary time overhead specific to our architecture. By integrating these with the existing fermionic SWAP (fSWAP) technique, we develop an efficient Trotter-based time-evolution operation for the 2D Hubbard model. Our analysis reveals an acceleration of over 10 times compared to naive serial compilation. Based on this optimized compilation, we estimate the computational resources required for the quantum phase estimation of the 2D Hubbard model. For devices with a physical error rate of $p_{\text{phys}} = 10^{-4}$, we estimate that approximately 6.2×10^4 physical qubits are required to achieve faster ground-state energy estimation of the 8×8 Hubbard model compared to classical computation.

DOI: [10.1103/93zr-lykb](https://doi.org/10.1103/93zr-lykb)

I. INTRODUCTION

Quantum computers are expected to provide exponential speedup over classical computers for various tasks, such as prime factoring [1], simulation of quantum many-body systems [2,3], and performing linear algebraic operations [4]. Recent advancements have led to the development of so-called noisy intermediate-scale quantum (NISQ) devices [5], which feature hundreds of qubits. Despite this progress, extracting practical quantum advantage from

such NISQ devices remains challenging owing to the existence of quantum noise.

The ultimate solution to quantum noise in a scalable manner is fault-tolerant quantum computing (FTQC) using quantum error-correction (QEC) codes. One promising FTQC architecture is based on the surface code [6–10], which boasts a high noise threshold and good compatibility with the superconducting qubit devices. These advantages have spurred several small-scale surface-code experiments [11–13]. However, resource estimations for several tasks indicate that surface-code-based FTQC architecture requires around 10^6 physical qubits to achieve meaningful quantum advantage [14–17]. This is several orders of magnitude more than what current NISQ devices, with hundreds of physical qubits, can offer. To bridge this gap, researchers are exploring ways to reduce the resource overhead necessary for the practical quantum advantage.

*Contact author: akahoshi.yutaro@fujitsu.com

Published by the American Physical Society under the terms of the [Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/) license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

Efforts include developing new algorithms with lower resource requirements [18–21], converting QEC overhead to other forms (such as a sampling overhead) through mitigation techniques [22,23], and developing new QEC codes that have smaller resource overhead than the surface code [24–30].

Under these circumstances, we have previously introduced the “space-time efficient analog rotation quantum computing architecture” (STAR architecture) [31], designed for the early stages of fault-tolerant quantum computing (early FTQC). This architecture integrates fault-tolerant Clifford gates, protected by QEC and performed via surface-code lattice surgery [10,32], with noisy analog rotation gates performed directly using a gate teleportation circuit consuming an ancilla state. By avoiding magic state distillation [33] and Solovay-Kitaev decomposition [34], the STAR architecture reduces the space-time overhead of non-Clifford gates. As a drawback, ancilla-state injection for the analog rotation gate is not fault tolerant; thus, we need to remove the errors occurring during state injection as much as possible. Recently, we have developed a novel ancilla-state injection protocol [35], inspired by Choi *et al.* [36]. This new protocol only works for small rotation angles θ . Its worst-case logical error rate is $\propto \theta p_{\text{phys}}$, where p_{phys} is the physical error rate. This is a factor of θ better than the original protocol of the STAR architecture [31], the behavior of which is $\propto p_{\text{phys}}$. In principle, this new protocol achieves sufficient precision to handle some important tasks in the early FTQC era, such as quantum simulations of condensed-matter physics.

For the practical application of the STAR architecture, the appropriate compilation of the input logical quantum circuit is mandatory. In the early FTQC era, where resources are limited, reducing unnecessary run-time overhead is significant in achieving quantum acceleration. There have been several discussions on circuit compilation in the context of the surface-code-based FTQC architecture [37–42], where the main overhead comes from the lattice-surgery operations that need additional ancilla logical qubits. If the number of available ancilla logical qubits is restricted, waiting for them to become available creates unnecessary overhead. Optimizing lattice-surgery operations is generally NP-hard [37] but several heuristic optimization techniques have been proposed [38–42]. In the STAR architecture, the situation differs owing to the use of the analog rotation gates. A major advantage of the STAR architecture is that analog rotation gates can easily be performed in parallel without any space overhead, due to the absence of distillation blocks. Compilers should utilize this advantage to reduce run-time. On the other hand, the run-time overhead of the STAR architecture also depends on the repeat-until-success (RUS) protocol and the ancilla-state injection protocol. The nondeterministic nature of the RUS protocol complicates precompiling its execution order. If ancilla-state injections rarely succeed,

the run-time per each RUS trial increases, and the success rate of state injections decreases with longer RUS protocol durations owing to increasing rotation angles. Therefore, to maximize the benefits of STAR architecture, a “real-time” strategy is needed to minimize the run-time overhead of the rotation gates, thereby suppressing the worst-case overhead of long-lasting RUS protocols.

In this paper, we present a method to compile a representative task while reducing the overhead of the RUS protocols. We focus on the quantum simulation of the two-dimensional (2D) Hubbard model [43], expected to be the benchmark of practical quantum advantage in the near future [15,44,45]. To utilize the small-angle rotation gate in the STAR architecture, we employ the Trotterization approach for the quantum simulations. We reduce the time overhead of the RUS protocol by employing the fermionic SWAP (fSWAP) network [46], which localizes the interaction terms of the Hamiltonian, allowing parallel execution of these local rotation gates. Additionally, we introduce two ideas to decrease the ancilla-state injection overhead: parallel injection and adaptive injection-region updating. Numerical simulations of the RUS protocol using these techniques indicate an 80% reduction in time overhead when the injection success rate is low. Combining all these technical elements, we derive a formula for the execution time of the time-evolution operator, which can be applied to any estimation of any computational task using Trotter-based time evolution of the 2D Hubbard model. As a result, we achieve an N times faster run-time for the $N \times N$ Hubbard model than the standard serial execution of the rotation gates [32], which is employed in Ref. [35]. To demonstrate the application of the formula, we estimate the total run-time and required number of physical qubits for the quantum phase-estimation (QPE) algorithm designed for the early FTQC era [21]. Using parameters from previous studies, we estimate that for a physical error rate of $p_{\text{phys}} = 10^{-4}$, an 8×8 Hubbard-model simulation can be performed in about 10 h with 6.2×10^4 physical qubits. This is approximately 10^3 times faster than the classical algorithm, given in Ref. [15], suggesting that the STAR architecture could offer practical quantum advantage with a smaller quantum device than previously expected.

This paper is organized as follows. In Sec. II, we provide a brief introduction to the STAR architecture, which forms the foundation of our discussion. In Sec. III, the main contribution of this paper, we explore an efficient operation schedule of the Trotter-based time-evolution operator within the STAR architecture. We start by introducing the Trotterization of the time-evolution operator and the Hamiltonians considered in this study. Key elements of our compilation, namely the fSWAP and parallel RUS execution, are discussed in detail. We then demonstrate how to estimate the total execution time of a single Trotter step, measured in units of the lattice-surgery-operation clock. In Sec. IV, we apply our compilation results to the QPE

algorithm. We estimate the execution time and the number of physical qubits based on our findings and the proposed algorithm [21]. Finally, in Sec. V, we summarize our study and briefly discuss some issues to be addressed in future studies.

II. OVERVIEW OF STAR ARCHITECTURE

Before delving into the detailed discussion on compilation, it is essential to outline how we perform universal quantum computing. Given our unique approach, we first review its features for readers who may be unfamiliar with it.

In this paper, we employ the STAR architecture [31] as the foundation of universal quantum computing. The key features of this architecture are the following:

- (1) fault-tolerant Clifford gates using QEC
- (2) low-overhead analog rotation gates with reasonably clean ancilla states
- (3) mitigating remnant noise in the analog rotation gate by quantum error-mitigation (QEM) techniques [e.g. probabilistic error cancellation (PEC)]

By performing the analog rotation gates directly, we can avoid the need for magic state distillation [33] and Solovay-Kitaev decomposition [34]. This significantly reduces the number of physical qubits and execution time of non-Clifford gates.

In this study, we employ the rotated planar surface code [10] as a logical qubit, also referred to as a “patch,” and use lattice surgery [10,32] to implement logical Clifford gates. Detailed descriptions of the logical Clifford operations used in this study are summarized in Appendix A. Regarding the low-overhead analog rotation gate, we employ the RUS approach [16,31,47] using the circuit in Fig. 1. In general, the injection of the ancilla state, $|m_\theta\rangle \equiv R_Z(\theta)|+\rangle = \frac{1}{\sqrt{2}}(e^{-i\theta}|0\rangle + e^{+i\theta}|1\rangle)$, cannot be fault tolerant owing to the analog rotation angle. To reduce noise, we employ a recently proposed ancilla-state injection protocol [35]. We briefly explain the RUS protocol and the state injection protocol in Appendix B. This protocol, although restricted to small rotation angles, drastically improves the logical error rate of the ancilla state. Since the Trotter-based time-evolution operator is a product of small angle rotation gates, this protocol is highly suitable for our purposes. As discussed in Ref. [35], by assuming the circuit-level noise model with error probability p_{phys} , the noise channel of the direct small-angle rotation gate can be reduced to a probabilistic Z error channel, the worst-case logical error rate of which is given as

$$\epsilon_{\text{RUS},\theta^*} \approx \alpha_{\text{RUS}} \theta^* p_{\text{phys}}, \quad (1)$$

where α_{RUS} is a factor that accounts for noise accumulation during RUS trials. α_{RUS} depends not only on the

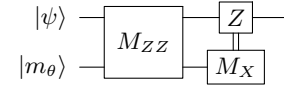


FIG. 1. The quantum circuit to implement $R_Z(\theta)$. M_{ZZ} and M_X represent the measurements of ZZ and X , respectively. Based on the measurement value m_{ZZ} of M_{ZZ} , the output state is either $R_Z(\theta)|\psi\rangle$ ($m_{ZZ} = +1$) or $R_Z(-\theta)|\psi\rangle$ ($m_{ZZ} = -1$). The circuit is repeated until the desired state $R_Z(\theta)|\psi\rangle$ is obtained.

rotation angle θ^* but also on the parameters chosen for the injection protocol. Numerical simulations [35] suggest that $\alpha_{\text{RUS}}/k \approx 0.40$ (where k is a parameter of the injection protocol; see Appendix B) and we use this value in our discussion. Once the noise channel is specified, we can mitigate it by QEC techniques. In this study, we employ PEC [48,49], which allows us to compensate the noise channel with an additional sampling overhead,

$$\gamma^2(\theta^*) \approx e^{4\epsilon_{\text{RUS},\theta^*}}. \quad (2)$$

In Sec. IV, we use this equation to estimate the actual run-time of the quantum algorithm, including the overhead introduced by PEC.

III. COMPILATION OF THE TROTTER-BASED TIME EVOLUTION OF 2D HUBBARD MODEL FOR STAR ARCHITECTURE

In this section, we discuss the compilation of the Trotter-based time evolution of the 2D Hubbard model, tailored for the STAR architecture. We begin by introducing the Trotter decomposition of the time-evolution operator and defining the 2D Hubbard-model Hamiltonian. Next, we outline our compilation strategy, emphasizing the core techniques used. After that, we introduce core techniques in our compilation. These include the insertion of an fSWAP network and the execution of the parallel RUS protocol. Finally, we sum up all these elements to determine the execution time of the single Trotter step. We also compare our compilation results with a naive sequential-compilation approach.

A. Trotter-based time-evolution operator

Consider the Hamiltonian given as

$$H = \sum_{j=0}^{L-1} c_j P_j, \quad (3)$$

where P_j represents a multi-qubit Pauli operator, c_j denotes a real coefficient, and L is the total number of terms. We aim to perform the time-evolution operator $e^{-iH\tau}$, but directly exponentiating the Hamiltonian is impractical owing to its exponentially large dimension.

To simplify this, we use the Trotter decomposition, which approximates the time-evolution operator with easy-to-perform steps, introducing a small error. We divide the time interval τ into r steps and define $\Delta\tau = \tau/r$. The second-order Trotter decomposition is given as

$$e^{-iH\Delta\tau} \approx \prod_{j=0}^{L-1} e^{-ic_j P_j \Delta\tau/2} \prod_{j=L-1}^0 e^{-ic_j P_j \Delta\tau/2} \equiv e^{-iH_{\text{eff}}\Delta\tau}, \quad (4)$$

where $\Delta\tau$ is sufficiently small. The approximation error of the second-order Trotterization behaves as [45],

$$\|e^{-iH\Delta\tau} - e^{-iH_{\text{eff}}\Delta\tau}\| \leq W\Delta\tau^3, \quad (5)$$

where $\|\cdot\|$ is the operator norm and W is a constant called the “Trotter error norm” [45]. The error of the n th energy eigenvalue $E_{\text{eff},n}$ calculated by $e^{-iH_{\text{eff}}\Delta\tau}$ is also bounded as [45]

$$|E_n - E_{\text{eff},n}| \leq W\Delta\tau^2. \quad (6)$$

The time evolution of $e^{-iH\tau}$ can be approximated by the sequence of the time evolution of $e^{-iH\Delta\tau}$,

$$e^{-iH\tau} = \left(\prod_{j=0}^{L-1} e^{-ic_j P_j \Delta\tau/2} \prod_{j=L-1}^0 e^{-ic_j P_j \Delta\tau/2} \right)^r + \mathcal{O}(1/r^2). \quad (7)$$

As shown, the Trotter-based time evolution is a product of small-angle Pauli rotations. In Sec. IV, Eq. (6) is used to determine the division number r for the Trotter decomposition via $\Delta\tau = \tau/r$.

B. Hamiltonian

In this study, we consider the 2D Hubbard-model Hamiltonian on an $N \times N$ square lattice. The 2D Hubbard model is a simple model that captures the electronic and magnetic behavior in solid materials [50,51]. The Hamiltonian of the 2D Hubbard model is defined as

$$H = -t \sum_{\langle i,j \rangle, \sigma} (c_{i,\sigma}^\dagger c_{j,\sigma} + \text{h.c.}) + U \sum_i n_{i,\uparrow} n_{i,\downarrow}, \quad (8)$$

where $c_{i,\sigma}^{(\dagger)}$ ($\sigma = \uparrow, \downarrow$) represents the creation (annihilation) operator of the fermion and $n_{i,\sigma} = c_{i,\sigma}^\dagger c_{i,\sigma}$ ($\sigma = \uparrow, \downarrow$) denotes the number operator of the spin-up and spin-down components on site i . The notation $\langle i,j \rangle$ indicates pairs of adjacent sites on the 2D lattice, while t and U are parameters indicating the hopping strength and repulsive potential

strength, respectively. The Jordan-Wigner (JW) transformation [52] with a certain index ordering transforms the Hamiltonian as follows:

$$H = -\frac{t}{2} \sum_{\langle i,j \rangle, \sigma} (X_{i,\sigma} X_{j,\sigma} + Y_{i,\sigma} Y_{j,\sigma}) Z_{i,j,\sigma}^{\leftrightarrow} + \frac{U}{4} \sum_i Z_{i,\uparrow} Z_{i,\downarrow} - \frac{U}{4} \sum_{i,\sigma} Z_{i,\sigma}, \quad (9)$$

$$Z_{i,j,\sigma}^{\leftrightarrow} = \prod_{k \in P_{ij}} Z_{k,\sigma}, \quad (10)$$

where P_{ij} indicates a set of sites between i and j in the one-dimensional (1D) ordering of the JW transformation. The ordering we choose in this study will be discussed later. Originally, the Hamiltonian consists only of local interaction terms. However, the JW transformation introduces nonlocal multi-qubit Pauli operators to maintain the fermionic commutation relations in the 1D mapping. As discussed in Ref. [44], the last term of Eq. (9) is proportional to the total electron-number operator up to some constant factor. Generally, the simulation is performed in a subspace with a fixed number of electrons, allowing us to omit the last term of Eq. (9) by absorbing the constant factor into the energy eigenvalues. Therefore, the Hamiltonian we consider is reduced to

$$H = -\frac{t}{2} \sum_{\langle i,j \rangle, \sigma} (X_{i,\sigma} X_{j,\sigma} + Y_{i,\sigma} Y_{j,\sigma}) Z_{i,j,\sigma}^{\leftrightarrow} + \frac{U}{4} \sum_i Z_{i,\uparrow} Z_{i,\downarrow}. \quad (11)$$

In this study, we apply open boundary conditions. The parameters $t = 1$ and $U = 4$ are chosen, which is consistent with previous studies [15,45].

C. Compile strategy

In the previous section, we have noted that the JW transformation results in nonlocal multi-qubit Pauli terms such as $P = XZ \dots ZX$ in the Hamiltonian, making these interaction terms challenging to execute in parallel. However, to reduce the run-time overhead of the RUS protocol of rotation gates, it is better to perform these gates in parallel. Unlike the existing FTQC architecture, the STAR architecture can dynamically prepare the ancilla states as needed without any ancilla-state factory. This capability allows for parallel execution of rotation gates without increasing qubit overhead. According to this observation, we consider the following compile strategy in this study:

- (1) localizing as many interaction terms as possible in terms of the Pauli operator
- (2) performing multiple local interaction terms in parallel using small-angle rotation gates

- (3) optimizing the run-time overhead by fine-tuning patch arrangements and lattice-surgery-operation schedules

We achieve the first objective by inserting fSWAP operations [46], which rearrange the JW transformation assignments to make subsets of interaction terms local. Using the rotation gates of the STAR architecture, we then perform multiple local interaction terms in parallel. Proper patch arrangements are important to maintain the parallelism and avoid disruptions from the lattice-surgery routing. Additionally, we mitigate the overhead of the state injection in the RUS protocol through parallel injection and adaptive injection-region updating. In the following, we will delve into each technical component before consolidating these techniques to present the final compilation results.

D. Fermionic SWAP insertion

To perform quantum many-body simulations on quantum computers, we need to map fermions to qubit states while preserving fermionic statistics. The typical approach for this mapping is the JW transformation. However, as seen in Eq. (9), this transformation generally disrupts the locality of interaction terms. The fSWAP operation [46] offers a solution by enabling all interaction terms in the Hamiltonian to be performed locally at the cost of additional operations. The basic idea of this method is simple: changing the JW transformation mapping to position all interacting pairs of spin orbits adjacently. The fSWAP operator acting on the pair of spin orbits p and q is defined as follows:

$$\text{fSWAP}_{p,q} = 1 + c_p^\dagger c_q + c_q^\dagger c_p - c_p^\dagger c_p - c_q^\dagger c_q. \quad (12)$$

This operator acts on the fermionic creation and annihilation operators as

$$\text{fSWAP}_{p,q} c_p^\dagger \text{fSWAP}_{p,q}^\dagger = c_q^\dagger, \quad (13)$$

$$\text{fSWAP}_{p,q} c_p \text{fSWAP}_{p,q}^\dagger = c_q. \quad (14)$$

Therefore, the fSWAP operator interchanges two spin orbits while retaining Fermi statistics. In this study, we only apply the adjacent fSWAP operation, namely $q = p + 1$. Using the JW transformation, we obtain a simple matrix form of the adjacent fSWAP operator,

$$\text{fSWAP}_{p,p+1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad (15)$$

which is just a sequence consisting of the standard SWAP gate and the controlled-Z (CZ) gate.

The key challenge lies in inserting the fSWAP gates to achieve fully local interactions while minimizing time overhead (circuit depth). Previous studies have discussed this issue and in this study, we employ the strategy in Ref. [53]. We illustrate this with a 4×4 Hubbard model, as depicted in Fig. 2. We consider two different qubit orderings of the JW transformations, the union of them covering all interaction terms [the orderings shown in Figs. 2(a) and 2(b)]. We start with the ordering shown in Fig. 2(a), where half of the hopping interactions can be performed locally since they are already adjacent. After performing these adjacent interactions, we perform fSWAP operations to transform the current ordering into the other one [the ordering shown in Fig. 2(b)]. According to Theorem 6 of

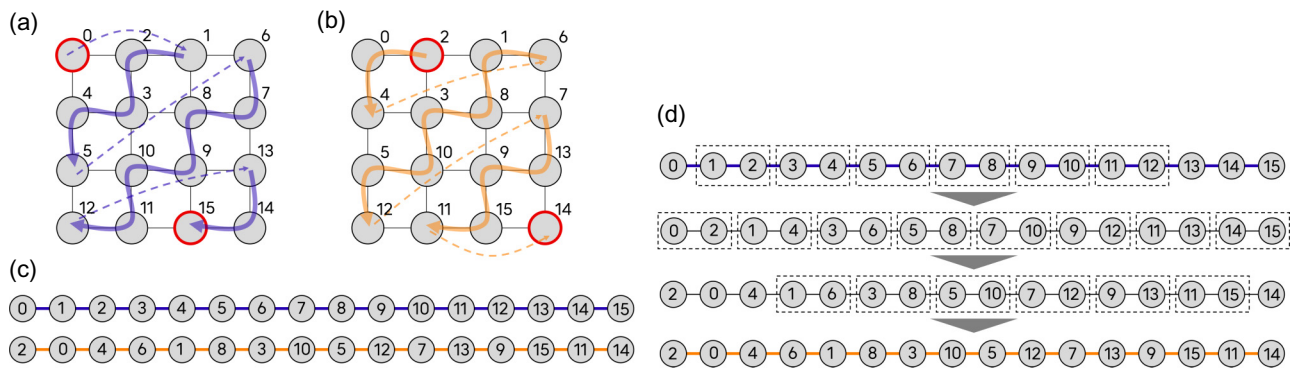


FIG. 2. A summary of fSWAP strategy used in this study, demonstrated by a 4×4 Hubbard model. (a),(b) The JW ordering considered in this study (for more details, see the text). We show only the up-spin components here, but the same orderings are also used in down-spin components. The beginning and end of the ordering are marked by red circles. The union of these two orderings covers all hopping-interaction edges. (c) A one-dimensional (1D) representation of the JW ordering from (a) and (b), with indices inserted from (a) to visualize the movement of each qubit via the fSWAP operations. (d) The adjacent fSWAP operations required to change the JW ordering from (a) to (b), with each layer performing fSWAP operations on pairs of qubits encircled by dashed lines. It takes three layers of fSWAP operations in this case.

Ref. [53], this transformation requires $N - 1$ fSWAP layers for the $N \times N$ Hubbard model. In Fig. 2(d), we show the explicit fSWAP operations needed to transform the ordering shown in Fig. 2(a) to that shown in Fig. 2(b). Finally, we perform the remaining half of the hopping interactions in the same manner as in the ordering shown in Fig. 2(a). The details of the lattice-surgery implementation will be discussed in Sec. III F.

E. Parallel RUS execution

Once hopping interactions become local, the STAR architecture can easily execute them in parallel. In this section, we will discuss the average number of RUS trials required for parallel execution and its advantages over serial RUS execution. We will also cover strategies to reduce the run-time overhead of the injection protocol, assuming a basic understanding of lattice surgery (details are provided in Appendix A).

Let us consider the case in which we perform M independent rotation gates in parallel. To estimate the average number of RUS trials needed to complete all M processes, we first calculate the probability that parallel RUS execution will finish on the K th RUS trial, P_K^M and we then calculate the expectation value of K . As derived in Appendix C, the expectation value of K behaves as

$$\begin{aligned} \langle K \rangle_M &= \sum_K K P_K^M \\ &= \sum_K K [(1 - 2^{-K})^M - (1 - 2^{-(K+1)})^M]. \end{aligned} \quad (16)$$

One can easily verify that if $M = 1$, we can reproduce $\langle K \rangle_{M=1} = \sum_K K/2^K = 2$. We plot $\langle K \rangle_M$ for different values of M in Fig. 3. As can be seen, the average number of RUS trials behaves in log-like fashion for large M ; therefore, parallel execution is drastically faster than serial execution if the injection protocol overhead is negligible.

In practice, however, the injection overhead can be significant in some cases. Therefore, let us discuss strategies to reduce the injection overhead and maximize the benefits of the parallelism. As discussed in Sec. II, the injection protocol is nondeterministic and can result in large time overhead when it fails repeatedly. Moreover, longer RUS protocols lead to larger rotation angles (for the K th trial, the rotation angle is 2^K times larger than the original), reducing the success rate since it depends on the rotation angle via p_{ideal} . Fortunately, the injection protocol uses only one logical patch to inject an ancilla state $|m_\theta\rangle$. This allows parallel execution if there are free patches available to connect to the logical patch where the rotation gate is to be performed. We show one such example in Fig. 4(a). We can also preinject the ancilla state needed for the next RUS trial in advance during the ZZ measurement of the current RUS trial (see Fig. 1) using available ancilla patches.

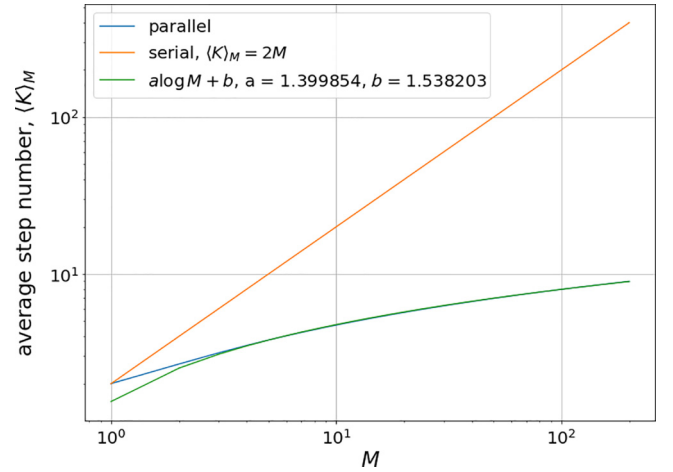


FIG. 3. The average number of RUS trials for M parallel RUS executions (blue line). The figure also shows the average trial number of serial RUS executions and a fitting result for parallel RUS execution. When M is large, the scaling of the parallel RUS execution is well described by the log function.

This “space” parallelism exponentially reduces the injection time overhead: if l free ancilla patches are available, the failure rate improves from P_{fail} to P_{fail}^l . Furthermore, we can heap up the injection protocol along the “time” direction, performing several trials of the injection within a single lattice-surgery clock cycle [see Fig. 4(b)]. This increases the number of injection trials further. Considering “space” and “time” parallelism strategies significantly minimizes the injection overhead until the rotation gate is executed.

The remaining challenge is how to assign free ancilla patches to the ongoing RUS processes. This assignment needs to be dynamic, adjusting as new free ancilla patches become available. To achieve this, we introduce the assignment algorithm based on a simple cluster-growing method, as illustrated in Fig. 5. This algorithm evenly distributes free patches among ongoing RUS processes. Due to this adaptive injection-region updating algorithm, we can suppress the worst-case run-time overhead caused from the long-lasting RUS processes and their low success rates owing to large rotation angles. Since long-lasting RUS processes are rare, most RUS processes in parallel execution will have already completed, leaving many free ancilla patches available for assignment to the remaining processes. As a result, massively parallelized state injection compensates for its low success rate. Numerical simulations support this scenario, showing that combining the parallel injection with the adaptive assignment algorithm effectively reduces the run-time overhead of parallel RUS execution (see the discussion of Sec. III G and Fig. 15).

At the end of this section, we comment on the validity of focusing on the average run-time of parallel RUS execution in this work. In general, the actual run-time of the

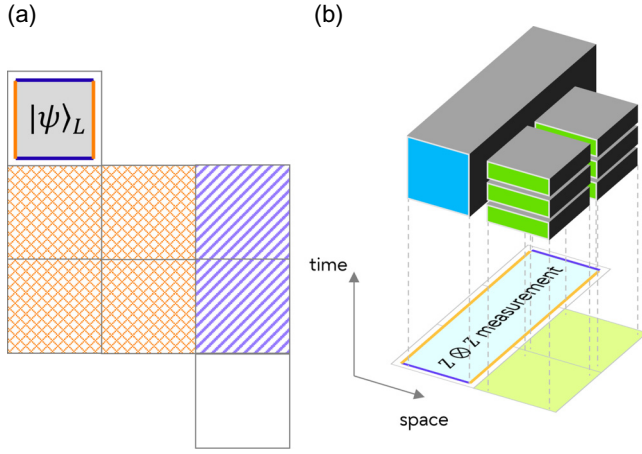


FIG. 4. An example of parallel injection. (a) An example of a free ancilla region that can be used for parallel injection. We consider applying a rotation gate on the logical patch $|\psi\rangle$, while the purple-shaded patches are already used in other operations. In this case, four orange-shaded patches can be used for the injection. The white patch cannot be used for the injection because there is no path connecting the target logical patch with this white patch. (b) Parallel injection along the time direction. During the ZZ measurement in the RUS circuit, we perform state injection for the next RUS trial on free patches in the injection region. Since the injection trial can be performed in a shorter time than a single lattice-surgery clock, we can repeat the injection several times (in this figure, three trials). If certain trials succeed, we can immediately execute the next RUS trial.

RUS execution sometimes deviates from the average value due to rare long-lasting events. However, the average run-time is sufficient for the run-time estimation in the early FTQC applications considered in our study. Let us consider the total run-time of the N_{RUS} different RUS executions, where each RUS run-time is T_i ($i = 1, 2, \dots, N_{\text{RUS}}$). The total run-time can be written as $T_{\text{RUS}} = \sum_i T_i = N_{\text{RUS}} \times \frac{1}{N_{\text{RUS}}} \sum_i T_i$; it is proportional to the sample average of T_i . In our run-time estimation, the situation is the same up to unimportant factors and N_{RUS} is large due to the sampling-based phase-estimation algorithm and the Trotter-based time evolution (it is almost identical to N_{total} in Table II). Therefore, the sample average sufficiently approaches the true average value; in other words, the actual run-time has negligible deviation from the estimation using the true average value [54].

F. Lattice-surgery implementations

In this section, we summarize the lattice-surgery implementations needed for the Trotter-based time evolution. We begin by introducing a suitable logical patch arrangement and then discuss lattice-surgery implementations for

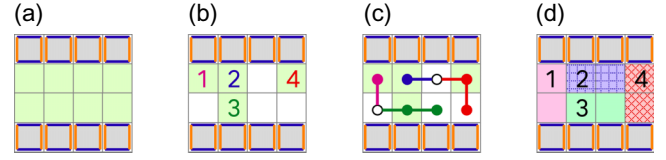


FIG. 5. A schematic description of the updating algorithm for the injection region during parallel Z rotations involving eight patches. (a) The initial injection region. Each RUS process has a single injection patch (green) adjacent to the target patch (gray). (b) After several RUS trials, four RUS processes complete and the corresponding injection regions become free (white patches). The updating algorithm then assigns these free patches to the ongoing four RUS processes (numbered 1–4). (c) To update the injection region for the ongoing processes, the algorithm expands the region clusters until the cluster can no longer grow, by adding adjacent free nodes step by step. There are two types of boundaries in the cluster: isolated (filled circles) and conflicted (open circles). A conflicted node means that different two region clusters add it at the same cluster-growing step. (d) The final result of the update algorithm. Isolated nodes can be assigned immediately to the respective injection regions. The conflicted nodes should be assigned to a smaller injection region (here, the size of the injection region is defined as the number of patches in the region) to balance the size of each injection region. As a result, each injection region becomes size 2 in this example.

each component of the Trotter-based time-evolution operator of the 2D Hubbard model. The basic operations of lattice surgery and logical Clifford gates used in this section are summarized in Appendix A.

As mentioned in Sec. III C, the logical patch arrangement is important to avoid unnecessary overhead from routing lattice-surgery operations. For our purpose, the local interaction terms and the fSWAP operations are nearest-neighbor two-qubit operations. Therefore, there is no need for a large routing region to perform these lattice-surgery operations. Additionally, the most complicated parts, the hopping interactions, are performed independently for each spin component, suggesting that dividing the logical patch arrangement into two regions that can operate separately is beneficial. Considering these requirements, we propose the logical patch arrangement shown in Fig. 6. This setup uses $2n$ logical patches to allocate n logical data qubits. The logical data qubits are arranged at the upper and lower sides, while the middle region is reserved for routing lattice-surgery operations. Each logical data qubit has an adjacent ancilla patch, allowing for single-qubit rotation gates on all logical qubits simultaneously. As discussed later, the upper-side (lower-side) logical qubits mostly carry the spin-up (-down) components, except for the execution of ZZ rotation terms.

Next, let us discuss how to perform the time-evolution operator in lattice surgery term by term. We begin with the implementation of the ZZ rotation terms, which are



FIG. 6. The patch arrangement considered in this study. Gray patches indicate logical data qubits. The white patches in the middle are used for lattice-surgery routing or ancilla-state injection.

part of the second term in Eq. (11). The Pauli ZZ operators in this term act on the spin-up and -down components on the same lattice site. To perform these ZZ rotations locally, we assign the spin components to logical patches as in Fig. 7. In Fig. 8, we show the actual lattice-surgery operations needed for ZZ interactions. Initially, each RUS process has two patches for injection at first (Fig. 7). As previously discussed, this initial injection region will be dynamically updated by the assignment algorithm. The execution time of the parallel ZZ rotation is denoted as $T_{\text{RUS}}(V, ZZ)$ clocks. We define the average parallel RUS execution clocks as $T_{\text{RUS}}(M, P)$, where M and P are the number of RUS processes performed in parallel and the rotation basis Pauli operator, respectively, with $V = N^2$. After performing the ZZ rotations, we move on to the more complicated operations: hopping interactions and fSWAP operations. But before that, we rearrange the logical patches to split the spin-up and -down components into the

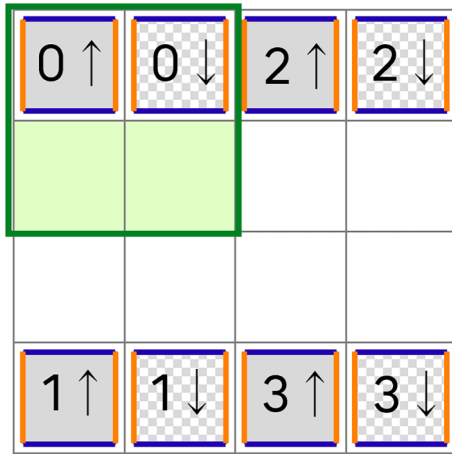


FIG. 7. The initial assignment of the logical information and execution of ZZ rotation terms. This figure shows only part of the whole patch arrangement here. Spin-up and spin-down components are placed on adjacent patches. ZZ rotation terms are then executed in parallel. The representative RUS process is indicated as a highlighted area by means of a green solid line and its initial injection region consists of green-colored patches.

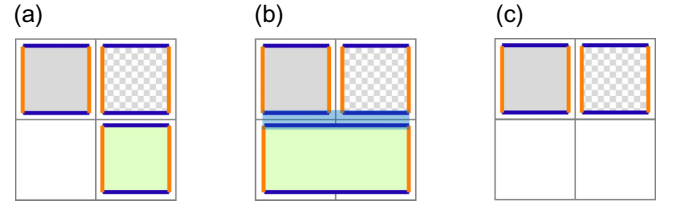


FIG. 8. Lattice-surgery implementation of the ZZ rotation gate. (a) The ancilla state $|m_\theta\rangle$ is prepared on a free patch (green patch). (b) To measure the ZZZ operator, the ancilla state is expanded to a rectangular shape and merged with the target two patches. (c) After splitting, the destructive logical X measurement is performed on the ancilla patch. The whole procedure takes 2 logical clocks.

upper and lower rows of the logical patches, respectively. This is necessary because the hopping terms are confined to each spin component and can be performed in parallel. This rearrangement is achieved by moving the logical patches, as shown in Fig. 9 [55]. This operation takes 3 clocks in total.

Next, let us move on to the lattice-surgery implementation of the hopping-interaction terms. We start by arranging the logical qubits in the first JW ordering, as shown in Fig. 2(a). This setup allows us to perform half of the hopping interactions via adjacent $XX + YY$ rotations. These interactions are further divided into two parts to avoid overlaps, as shown in Fig. 10. Each $XX + YY$ rotation is performed using the circuit in Fig. 11, with its lattice-surgery operation schematically represented in Fig. 12. The number of $XX + YY$ rotations performed at the same time is $N(N - 1)$, since each JW ordering contains $2N(N - 1)$ local hopping terms, and they are divided into horizontal and vertical. Therefore, the execution time of the $XX + YY$ rotations in the first JW ordering is: $2 \times (9 + T_{\text{RUS}}(N(N - 1), ZZ) + T_{\text{RUS}}(N(N - 1), Z))$.

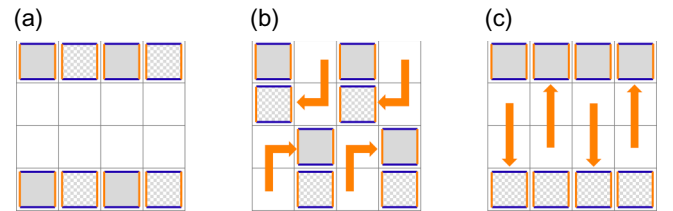


FIG. 9. Lattice-surgery operation of moving patches. The filled (hatched) patches carry spin-up (spin-down) components. (a) After completing the ZZ rotation gates, the patches are moved to split spin-up and spin-down components. (b) Half of the logical patches are moved to the middle region. This operation takes 2 lattice-surgery clocks. (c) Then, the logical patches in the middle region are moved to the upper or lower side, depending on the spin components that they carry. This takes 1 lattice-surgery clock. Therefore, the moving operation takes 3 logical clocks in total.

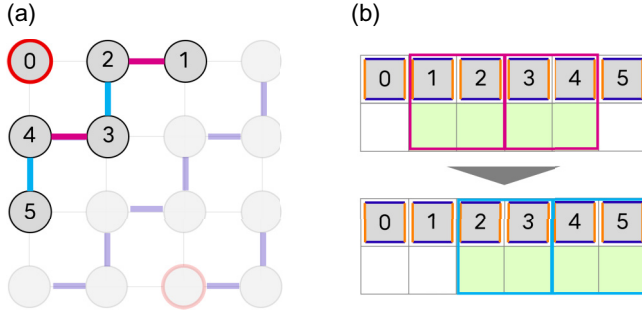


FIG. 10. An example of the parallel $XX + YY$ rotation. (a) A subset of the up-spin hopping-interaction terms of the 4×4 Hubbard model (see the densely shaded part), but other parts can be performed in the same way. The rotation gates are performed in two steps due to shared qubits: horizontal interactions (magenta edges) and vertical interactions (cyan edges). (b) The minimum RUS process as highlighted in the highlighted densely shaded area. Free patches at the beginning of the execution of the $XX + YY$ rotation [e.g. the leftmost white patches in (b)] are assigned to the neighboring RUS process to boost the injection protocol.

After completing the first half of the hopping terms, we switch to the other JW ordering using fswap operations. The logical fswap operation is realized through lattice-surgery operations in Fig. 13. We perform this logical fswap operation layer $N - 1$ times for the $N \times N$ Hubbard model. The logical fswap operation takes 7 clocks and therefore the fswap layer between two JW orderings takes $7(N - 1)$ clocks in total. Then, the remaining half of the hopping terms is performed similarly to the first JW ordering, taking $2 \times (9 + T_{\text{RUS}}(N(N - 1), ZZ) + T_{\text{RUS}}(N(N - 1), Z))$ clocks.

Finally, let us estimate the execution time of the second-order Trotter time evolution. The order of the operations to perform the second-order Trotter time evolution is summarized in Fig. 14. Gathering the execution clocks of all the components, we obtain

$$T_{\text{Trotter}} = 7T_{\text{RUS}}(V - N, Z) + 7T_{\text{RUS}}(V - N, ZZ) + 2T_{\text{RUS}}(V, ZZ) + 14N + 55. \quad (17)$$

The first two terms are derived from the RUS of $XX + YY$ rotation gates. Since the two $XX + YY$ rotation layers in the middle of the second-order Trotter formula can

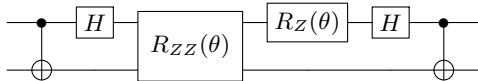


FIG. 11. The circuit to implement the $XX + YY$ rotation gate with a rotation angle θ . This circuit can be derived by the simultaneous diagonalization of the XX and YY bases [16] or by an equivalent transformation from the standard $XX + YY$ rotation circuit.

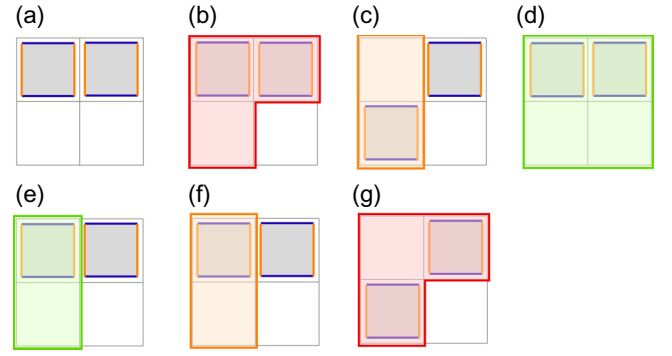


FIG. 12. Lattice-surgery operations of the $XX + YY$ rotation gate. Parts (a)–(g) are arranged by the order of the lattice-surgery operations. The logical CNOT gate and the logical Hadamard gate are detailed in Appendix A. (a) The initial configuration of the logical patches. (b) The first logical CNOT operation by using the highlighted red area. (c) The first logical Hadamard gate. The logical patch that has moved to the adjacent patch during the patch rotation does not need to move back to the original position. Therefore, this logical Hadamard gate is performed in 2 lattice-surgery clocks. (d) The RUS execution of the ZZ rotation gate. The initial injection region is shown as the green area. (e) The RUS execution of the single Z rotation gate. (f) The second logical Hadamard gate. Similarly to the first logical Hadamard gate, it takes only 2 lattice-surgery clocks. (g) The second logical CNOT gate. This operation takes 3 clocks due to the patch moving to the starting position. The entire operation takes $9 + T_{\text{RUS}}(M, ZZ) + T_{\text{RUS}}(M', Z)$ lattice-surgery clocks in total.

be merged into a single $XX + YY$ rotation layer, their coefficients become 7 instead of 8. By applying T_{Trotter} with an appropriate estimation of $T_{\text{RUS}}(M, P)$, we can estimate the run-time of quantum algorithms using the time-evolution operator in the STAR architecture. In Sec. IV, we demonstrate the application to the quantum phase-estimation algorithm and discuss the required resource in detail. Before delving into resource estimation, we discuss how to estimate $T_{\text{RUS}}(M, P)$ in the remainder of this section.

G. Estimation of $T_{\text{RUS}}(M, P)$

As discussed before, to estimate $T_{\text{RUS}}(M, P)$, we consider the QEC parameters such as the physical error rate, the configuration of parallel RUS executions, and how the injection region changes during the execution. Therefore, in general, $T_{\text{RUS}}(M, P)$ should be estimated by numerical simulations. In this study, we perform a two-step numerical simulation to estimate $T_{\text{RUS}}(M, P)$: ancilla-state injection and parallel RUS execution. The former simulates the success rate of the injection protocol under the circuit-level noise model with a physical error probability p_{phys} . This simulation uses a stabilizer formalism and we employ Stim

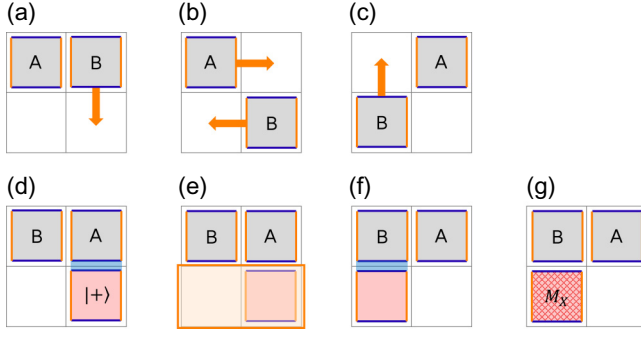


FIG. 13. Lattice-surgery implementation of the fSWAP gate. We perform the fSWAP gate by the sequence of the SWAP gate and the CZ gate. (a)–(c) The SWAP operation between patches A and B. (d)–(g) The logical CZ operation between patches A and B. (d) First, an ancilla patch is initialized to the $|+\rangle$ state and then the ZZ measurement is performed between patch A and the ancilla patch. (e) Then, the logical Hadamard gate acts on the ancilla patch. When the patch rotates, the ancilla patch moves to the adjacent position. (f) The ancilla patch and patch B are measured by the ZZ operator. (g) Finally, the ancilla patch is measured destructively by the X operator. In total, the fSWAP gate takes 7 lattice-surgery clocks.

[56] as the back end. Next, we simulate parallel RUS execution and the adaptive assignment algorithm based on the given injection success rate and the RUS configuration.

As a benchmark, we simulate 32 parallel small-angle Z rotations with the target angle $\theta = 10^{-3}$. The logical patch arrangement follows Fig. 6 and the initial injection region consists of a single adjacent patch for each RUS process, similar to Fig. 5(a). We compare execution times with a fixed injection region (naive) versus the space-time parallel-injection technique combined with the adaptive region updating (adaptive). We run 10^3 simulations for each case under two parameter sets: one corresponding to a low success rate ($p_{\text{phys}} = 10^{-3}$ and $d = 16$) and the other one corresponding to a high success rate ($p_{\text{phys}} = 10^{-4}$ and $d = 8$). The resultant histograms of these simulations are displayed in Fig. 15 (see the Supplemental Material [57]).

First, let us discuss the numerical results using the low success rate setup shown in Fig. 15(a). The long-lasting RUS processes, hindered by the low injection success rate, significantly degrade the performance of parallel execution. This is evident from the blue histogram in Fig. 15(a), which shows a long tail and an average run-time of about 130 clocks. The majority of this time, almost 94%, is consumed by the injection process, as the average RUS step count is less than 10 when $M = 32$, as shown in Fig. 3, and the ZZ measurement in the gate teleportation circuit takes only 1 clock. Conversely, the parallel-injection protocol and the adaptive assignment algorithm significantly reduce this overhead, as shown by the orange histogram in Fig. 15(a). The average run-time drops to about 25 clocks, achieving an 80% reduction in execution time. This result

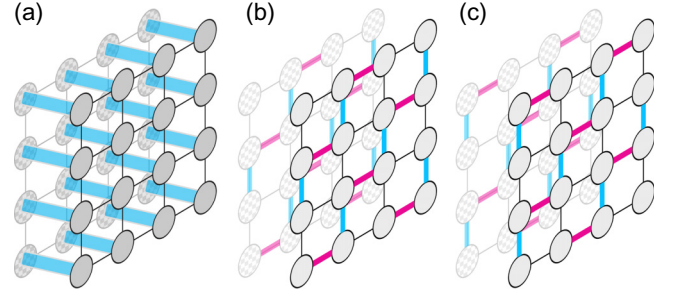


FIG. 14. The order of operations in the Trotter time evolution for the 4×4 Hubbard model. The interaction terms are performed in the order from (a) to (c). Spin-up components are shown as filled circles and spin-down components as hatched circles. This figure illustrates the first half of the second-order Trotter formula. The inverse-order operations follow after operation (c). (a) The inter-spin ZZ rotation terms, followed by the patch-moving layer. (b) The neighboring $XX + YY$ rotation terms in the first JW ordering. As discussed before, the interactions are divided into two subsets (magenta and cyan edges). After this operation, we perform fSWAP layers to move on to the second JW ordering. (c) The $XX + YY$ rotation terms in the second JW ordering.

confirms that parallel injection suppresses the injection overhead, while the adaptive assignment algorithm mitigates the overhead from the long-lasting RUS process. The very short tail in the orange histogram highlights the effectiveness of this cancellation, indicating that our adaptive parallel RUS execution is highly effective in scenarios with low injection rates.

For the high success rate setup, the naive implementation faces fewer issues, since injection trials rarely fail. However, our parallel injection and adaptive assignment algorithm improves the entire performance, as shown in the orange histogram in Fig. 15(b). While the parallel injection does improve the success rate, the primary benefit comes from “preinjection” for the next RUS trial during the ZZ measurement of the current RUS trial, as illustrated in Fig. 4(b). With injections almost always succeeding within 1 clock, this preinjection technique hides the injection overhead of subsequent RUS trials. As a result, the average clock approaches the theoretical lower-bound value of $M = 32$ (shown in Fig. 3). In summary, our parallel injection and adaptive assignment algorithm is valuable across a wide range of scenarios, for both low and high injection success rates.

If precise estimation of $T_{\text{RUS}}(M, P)$ is not essential, numerical simulations may not be the most efficient approach. Instead, one can use the $\langle K \rangle_M$ values shown in Fig. 3 together with some assumptions about the injection run-time. The assumed injection clock should be an upper bound of the actual injection run-time. For example, if the physical error rate is as low as $p_{\text{phys}} = 10^{-4}$, it is reasonable to assume that injection succeeds within 1

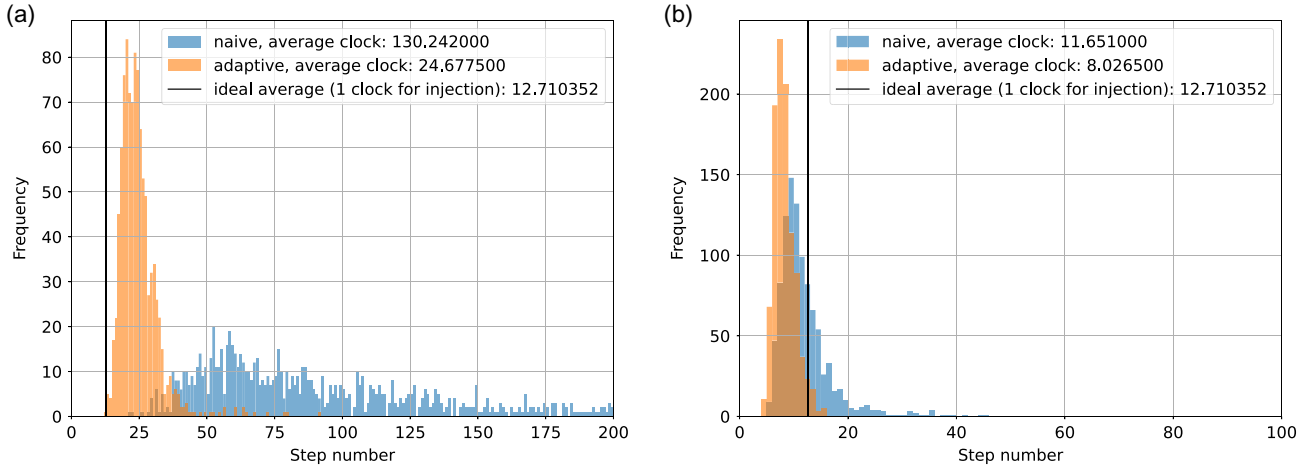


FIG. 15. Histograms of the total RUS steps to finish 32 parallel RUS processes. The horizontal axis shows the run-time taken to complete all RUS processes in units of the lattice-surgery clock, while the vertical axis shows the event counts. We also indicate the average run-time assuming that the injection succeeds with a single lattice-surgery clock. (a) The case of the physical error probability $p_{\text{phys}} = 10^{-3}$. (b) The case of the physical error probability $p_{\text{phys}} = 10^{-4}$.

clock, a fact supported by the numerical verification in Sec. IV B. Using this assumption, one can roughly estimate the average run-time of parallel Z rotations as follows:

$$T_{\text{RUS}}(M, P) \approx \langle K \rangle_M \times \{ (1 \text{ clock for injection}) + (1 \text{ clock for } Z \otimes P \text{ measurement}) \}. \quad (18)$$

By combining reasonable assumptions for the injection and Eqs. (17) and (18), one can easily estimate the execution time of Trotter-based algorithms using the STAR architecture.

H. Comparison with the serial-compilation strategy

Finally, let us compare our compilation result with a more standard approach in FTQC [58]. In this standard compilation, the Clifford operations are absorbed into the rotation gates and Pauli measurements [32], allowing any quantum circuit to be executed using multi-qubit Pauli rotation gates and measurements. Parallel execution of these multi-qubit Pauli rotations is generally challenging, so the rotation gates are executed sequentially (this compilation method is referred to as serial compilation). We discuss the details of this compilation method in Appendix F. In Fig. 16, we show the run-time of the second-order Trotter formula by using both parallel and serial compilations. The $T_{\text{RUS}}(M, P)$ in the parallel compilation is estimated based on parameters discussed in Sec. IV. As a result, we can achieve a significant reduction in run-time, ranging from 86% to 97% in this setup. Asymptotically, in the standard sequential-compilation method [58], the rotation gates in the Trotter formula are performed sequentially, taking $\Theta(N^2)$ lattice-surgery clock cycles for a single Trotter step [see Eq. (F2) in Appendix F]. In our

parallel-compilation method, on the other hand, according to Eqs. (17) and (18) and Fig. 3, a single Trotter step takes $\Theta(N)$ lattice-surgery clock cycles for sufficiently large N . In this sense, we can say that our compilation is N times faster than the sequential compilation.

IV. APPLICATION OF THE COMPILATION RESULT

In this section, we demonstrate how to apply our compilation results to possible early FTQC tasks, specifically the QPE algorithm. First, we introduce the QPE algorithm used in this study and estimate the resource requirements

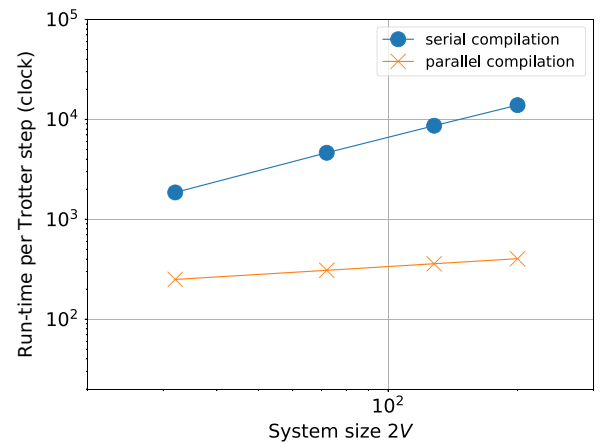


FIG. 16. Comparison with the serial-compilation result. The horizontal axis indicates the system size $2V = 2M^2$ for the $M \times M$ Hubbard model. The blue circles (orange crosses) indicate the run-time of a single Trotter step with the serial (parallel) compilation strategy.

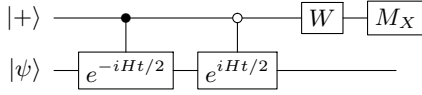


FIG. 17. The Hadamard test circuit. We set $W = I$ ($W = S^\dagger$) to calculate $\text{Re} \langle \psi | e^{-iHt} | \psi \rangle$ ($\text{Im} \langle \psi | e^{-iHt} | \psi \rangle$).

for the QPE of the 2D Hubbard model using our compilation results. While our estimation relies on empirical parameters from previous studies, we expect that the order of the estimated values will not change drastically. This demonstration will be useful for readers looking to estimate the resource requirements of certain algorithms within the STAR architecture.

A. QPE algorithm for early FTQC

Focusing on early FTQC devices, we employ a recently proposed QPE algorithm [21] designed for such devices, rather than traditional textbook wisdom [59]. This algorithm relies on the Hadamard test, as shown in Fig. 17. The Hadamard test circuit requires only a single ancilla qubit, making it suitable for early FTQC devices. To execute this circuit, we need to perform a controlled time-evolution operator. The ancilla state in Fig. 17 is placed next to the routing region (Fig. 18). In Appendix D, we discuss its implementation using our compilation results discussed above. By sampling and properly averaging the measurement results, we can extract the expectation value of the time evolution operator:

$$\langle \psi | e^{-iHt} | \psi \rangle = \sum_i p_i e^{-iE_i t}, \quad (19)$$

where $|\psi\rangle = \sum_i a_i |E_i\rangle$ is an input state, $|E_i\rangle$ is the i th eigenstate of the Hamiltonian, and $p_i = |a_i|^2$. As observed, this expectation value is a sum of the exponential factors of different eigenstates, allowing us to extract energy eigenvalues from its time dependence using classical signal processing. In this study, we employ the quantum complex exponential least-squares (QCELS) algorithm [21] to achieve this. This algorithm is preferable since it requires relatively lower circuit depth compared to other algorithms [21]. We refer to the main algorithm of the QCELS algorithm and the statement of its computational complexity as Algorithm 1 and Theorem E1 in Appendix E. The following resource estimation is based on these elements.

B. Resource estimation

We are now ready to discuss the resource-estimation details. First, we summarize some assumptions and parameter settings, such as the target precision. Based on these settings, we derive the required number of Trotter steps in the QCELS algorithm. By combining these results with

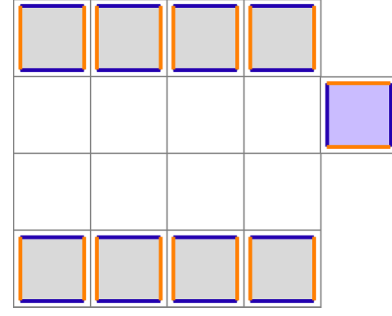


FIG. 18. The patch arrangement in the QPE algorithm. We show only a part of the entire patch arrangement. The blue patch denotes the ancilla qubit for the Hadamard test circuit.

our compilation results, we determine the code distance. After verifying this code distance through numerical simulations, we estimate the execution time and the number of physical qubits needed.

We consider the ground-state energy estimation with a certain precision ϵ_{targ} , $E_0 + \epsilon_{\text{targ}}$, of the $M \times M$ Hubbard model ($M = 4, 6, 8, 10$). The target precision of the QPE is set to $\epsilon_{\text{targ}} = 0.01$. This choice has been commonly used in previous studies [15,45]. Two algorithmic errors affect the final estimation: the Trotter error $\epsilon_{\text{Trotter}}$ and the estimation error of the QCELS, ϵ_{QCELS} . In the worst case, their linear combination contributes to the error of the final result, so they should satisfy

$$\epsilon_{\text{QCELS}} + \epsilon_{\text{Trotter}} \leq \epsilon_{\text{targ}}. \quad (20)$$

As seen in Theorem E1, the QCELS algorithm assumes that the target phase is in $[-\pi, \pi]$, so we normalize the Hamiltonian as follows:

$$\tilde{H} \equiv \frac{\pi}{\lambda} H, \quad \lambda \equiv \|H\|_1, \quad (21)$$

where $\|H\|_1 = \sum_j |c_j|$ is the sum of absolute values of the coefficients in the Hamiltonian [see Eq. (3)], usually referred to as the 1-norm of the Hamiltonian. Therefore, the terms in Eq. (20) are replaced as follows:

$$\tilde{\epsilon}_{\text{QCELS}} + \tilde{\epsilon}_{\text{Trotter}} \leq \tilde{\epsilon}_{\text{targ}}, \quad (22)$$

where the tilde denotes that it is normalized by the factor of π/λ . The λ values can be estimated for each $M \times M$ Hubbard model and are listed in Table I. The QCELS algorithm parameters are determined by the statement of Theorem E1: $J = \lceil \log_2(1/\tilde{\epsilon}_{\text{QCELS}}) \rceil + 1$, $\tau_j = 2^{j-J} \frac{\delta}{N \tilde{\epsilon}_{\text{QCELS}}}$, $\forall 1 \leq j \leq J$. The choice of δ is somewhat empirical. Theoretically, δ is chosen as $\delta = \Theta(\sqrt{1-p_0})$, where p_0 is the overlap factor between the input state and the ground state. However, numerical simulations in Ref. [21] indicate that δ can be smaller than the theoretical prospect, namely $\delta \approx 0.06$ with $p_0 = 0.8$ (note that

TABLE I. The λ values of our Hamiltonian.

	4×4	6×6	8×8	10×10
λ	64	156	288	460

$\sqrt{1 - 0.8} \approx 0.45$). We optimistically use this value in our estimation. We set the number of data pairs to $N = 5$ and the number of samples to $N_s = 100$. This choice is consistent with the numerical simulations shown in Ref. [21]. This parameter setting aligns with the leading behavior (except for the polylog factor) of the statement of Theorem E1 as $NN_s \approx 1/\delta^2$.

Next, let us discuss $\tilde{\epsilon}_{\text{Trotter}}$. In this estimation, we employ the second-order Trotter decomposition. We split each $\tau_j/2$ ($j = 1, 2, \dots, J$) into N_j Trotter steps (note that the factor of 2 comes from the Hadamard circuit in Fig. 17), $\Delta\tau_j = \tau_j/2N_j$. From Eq. (6), $\Delta\tau_j$ must satisfy

$$\tilde{W}\Delta\tau_j^2 \leq \tilde{\epsilon}_{\text{Trotter}}, \quad (23)$$

and therefore N_j should satisfy

$$N_j \geq \frac{\tau_j}{2} \sqrt{\frac{\tilde{W}}{\tilde{\epsilon}_{\text{Trotter}}}}, \quad (24)$$

where $\tilde{W}$ is a rescaled Trotter error norm defined as $\tilde{W} = (\pi/\lambda)^3 W$. The $\tilde{W}$ value should be estimated to determine N_j . We use the W value provided in Ref. [45] in our estimation and rescale it (although the details of the setup are different from those of Ref. [45], we expect that the value of W is similar).

By combining Eqs. (E4), (E5), and (24), the total number of Trotter steps is derived as follows:

$$N_{\text{total}} = \sum_{j=1}^J \frac{(N-1)N_s 2^{j-J} (\tilde{\epsilon}_{\text{QCELS}})^{-1} \delta}{\tilde{\epsilon}_{\text{QCELS}}} \sqrt{\frac{\tilde{W}}{\tilde{\epsilon}_{\text{Trotter}}}}. \quad (25)$$

We can minimize N_{total} by optimizing $\tilde{\epsilon}_{\text{QCELS}}$ and $\tilde{\epsilon}_{\text{Trotter}}$ under the condition of Eq. (22). Using the optimization results, we can calculate the maximal number of Trotter steps needed for a single Hadamard test circuit, $N_{\text{max}} = N \times N_J = \frac{\delta}{2\tilde{\epsilon}_{\text{QCELS}}} \sqrt{\frac{\tilde{W}}{\tilde{\epsilon}_{\text{Trotter}}}}$. These values are listed in Table II.

From this, we determine the required code distance in two steps: (i) we estimate the total number of logical operations, N_{op} , from N_{max} and T_{Trotter} [Eq. (17)], and then (ii) we determine the code distance that satisfies $p_{\text{logical}}(d) \times N_{\text{op}}(d) < \epsilon_{\text{logerr}}$, where p_{logical} is the logical error rate per single logical operation and ϵ_{logerr} is the required logical error rate in a single circuit run. In the following estimation, we employ $p_{\text{logical}} = 0.1d(100p_{\text{phys}})^{(d+1)/2}$ with the physical error rate $p_{\text{phys}} = 10^{-4}$, while ϵ_{logerr} is set to 0.01

TABLE II. The resultant N_{total} and N_{max} by the minimization of Eq. (25).

	4×4	6×6	8×8	10×10
N_{total} (Trotter step)	2 717 609	4 040 743	5 399 835	6 830 416
N_{max} (Trotter step)	3397	5051	6750	8538

[32]. The N_{op} depends on the code distance d via the success rate of ancilla-state injection in the RUS process. As discussed in Sec. III, the success rate and average execution time of RUS should be estimated numerically for precise results. However, because running numerous simulations for various code distances is inefficient, we first assume that the injection finishes within 1 clock and roughly determine the code distance. The assumption is expected to closely match the actual scenario, given the physical error rate $p_{\text{phys}} = 10^{-4}$. After determining the code distance, we numerically simulate parallel RUS execution. If the numerical run-time is smaller than the rough estimation, the chosen code distance is sufficient. The determined code distance and estimated run-time per Trotter step are summarized in Tables III and IV, respectively.

Once the code distance and average RUS execution time are determined, we can estimate the total execution time in seconds, assuming that each code cycle (= syndrome measurement) takes 1 μs [32]. To make our estimation more precise, we also account for the PEC overhead to compensate for the logical errors from the injection. By combining the second-order Trotter formula of Eq. (7) and the PEC overhead formulas of Eqs. (1) and (2), we calculate the sampling overhead of the Hadamard test with time evolution τ as

$$C_\tau \approx e^{4\alpha_{\text{RUS}}\pi\tau p_{\text{phys}}}. \quad (26)$$

We estimate the overhead of Eq. (26) for all Hadamard test circuits with $\{\tau = \tau_j n/2 \mid j = 1, \dots, J, n = 0, 1, \dots, N-1\}$ and include them in the total run-time estimation. The number of physical qubits is estimated as $2d^2$ for each surface-code patch. Since our patch arrangement needs $4M^2 + 1$ logical patches for $M \times M$ Hubbard-model simulation (+1 comes from the ancilla qubit of the Hadamard test circuit), the total number of physical qubits is given as $(4M^2 + 1) \times 2d^2$ for the $M \times M$ Hubbard model. These results are summarized in Table V (see the Supplemental Material [57]). In Fig. 19, we compare our total run-time estimation with that of a classical computer using the tensor-network algorithm [15]. Although the boundary conditions in the classical setup in Ref. [15] differ from ours, we expect the run-time order to be similar. Our rough estimation indicates the possibility of quantum speedup in the 8×8 Hubbard-model simulation, even with 6.2×10^4 physical qubits. We also see the potential in the 6×6

TABLE III. The verified code distance and parameter of the injection protocol.

	4×4	6×6	8×8	10×10
d	9	11	11	11
$ Q_i $	$[3, 3, 3] (k = 3)$	$[2, 2, 2, 2, 3] (k = 5)$	$[2, 2, 2, 2, 3] (k = 5)$	$[2, 2, 2, 2, 3] (k = 5)$

Hubbard model, although this is less certain owing to the overhead of state preparation in the QPE, which we have not considered in our estimation. For the 4×4 Hubbard model, the run-time is slower than the classical algorithm but it completes in about 2 h and requires only about 10^4 physical qubits. This setup is valuable for the early FTQC era to directly compare quantum and classical algorithm results, verifying the accuracy of the quantum computer. Given the recent progress in quantum computing hardware, such verification is expected to be possible in the near future. This result indicates that our architecture can serve as a good test bed for achieving quantum speedup in meaningful tasks during the early FTQC era.

C. Possibility of further reduction of resource requirement

Finally, we discuss the possibility of resource reduction from two aspects: the device and the Trotterization method.

The estimated total run-time assumes that a single quantum computer with over 10^4 physical qubits runs all the necessary Hadamard test circuits in series. However, since these circuit runs are independent, we can easily reduce the total run-time by running them in parallel using multiple quantum computers. For example, if we employ ten independent quantum computers, the possible total run-time is represented by the blue points in Fig. 19. The maximum run-time shown in Table V represents the theoretical lower bound achievable through a parallel sampling. Generally, once a quantum device with order of 10^4 physical qubits is available, producing ten such devices seems much easier than developing a single device with 10^5 physical qubits that operates coherently. Typical FTQC algorithms require more physical qubits to operate coherently to reduce run-time by increasing distillation blocks. Therefore, achieving quantum speedup based on our proposed parallel sampling approach is promising, even from the perspective of device development.

Next, let us comment on possible improvements to the Trotterization method. Our estimation is based on the theoretical Trotter error bound, but a previous study shows

that the empirical error bound can be much lower than that [60]. If this is true, we need fewer Trotter steps than our estimation here to achieve the same precision, resulting in the faster run-time. Further improvement can be achieved by some other advanced Trotterization methods, such as the randomization approach [61,62]. Previous studies show that it can reduce the gate count needed to simulate the time-evolution operator over the standard Trotterization approach by randomly sampling the rotation gates. A nontrivial point is that the structure of the time-evolution operator becomes random, making it more difficult to parallelize the rotation gates than the standard approach. Further investigations on which approach is the best for the STAR architecture are important and we believe that this paper provides a foundation for such investigations at the lattice-surgery level.

V. SUMMARY AND CONCLUSIONS

In this paper, we focus on simulating the 2D Hubbard model and discuss how to efficiently perform its time-evolution operator on the STAR architecture. We use the fSWAP gate to localize interaction terms, enabling parallel execution of local rotation gates. To reduce the overhead from ancilla-state injection, we combine two strategies: space-time parallel injection and adaptive updating of the injection region. These approaches result in an over 80% reduction in the execution time against the naive serial rotation gate scheme, especially when the injection success rate is low. Using the compilation results of the time-evolution operator, we estimate the resource requirements for the QPE algorithm designed for early FTQC devices. Our rough estimation indicates that we can achieve over 10^3 times quantum speedup for simulating the 8×8 Hubbard model using 6.2×10^4 physical qubits with $p_{\text{phys}} = 10^{-4}$. This is encouraging, as it suggests that realizing quantum speedup for practical tasks may happen sooner than previously expected, due to the STAR architecture.

Finally, we comment on some topics that we do not address in this paper:

- (i) *Other possible applications of the compilation result.* The compilation result we have obtained is a simple time-evolution operator, which can be applied to several tasks beyond the QPE. For example, the controlled time-evolution operator can be used for the imaginary time evolution [63], which is essential for probing the finite-temperature structure of the system and increasing the ground-state

TABLE IV. The average clock per Trotter step derived from numerical simulations.

	4×4	6×6	8×8	10×10
T_{Trotter} (clock)	250.03	308.48	358.95	403.32

TABLE V. The estimated run-time of the QPE algorithm and the required number of physical qubits with $p_{\text{phys}} = 10^{-4}$. The total run-time includes all sampling run times of the Hadamard test circuit accounting for the QEM overhead. “Maximum run-time” refers to the maximum run-time of the single run of the Hadamard test circuit, as estimated by N_{max} in Table II.

	4×4	6×6	8×8	10×10
Total run-time (s)	6934.43	18368.61	35194.71	67560.34
Maximum run-time (s)	7.64	17.14	26.65	37.88
N_{qubit}	10 530	35 090	62 194	97 042

overlap. In addition, we can estimate the run-time for variational *Ansätze* with similar structures, such as the variational Hamiltonian Ansatz (VHA) of the 2D Hubbard model. Combining the VHA with the local variational quantum compiling (LVQC) [64,65] can further reduce the circuit depth. Additionally, the run-time for the quantum-selected configuration interaction (QSCI) algorithm [66] can be estimated by treating the variational *Ansatz* as the adiabatic state-preparation circuit.

- (ii) *Compilation of other systems.* While this paper focuses on the 2D Hubbard-model Hamiltonian to show how we can achieve the quantum advantage in the near future, our compilation strategy can be applied to other systems. An important class of Hamiltonian is provided as [67]

$$H = \sum_{p,q} T_{pq} c_p^\dagger c_q + \sum_p U_p n_p + \sum_{p \neq q} V_{pq} n_p n_q. \quad (27)$$

This Hamiltonian contains a range of physical systems, such as molecules with a discretized space

cell and crystalline solids using the plane-wave basis [67]. Using the fSWAP network, we can achieve the Trotter decomposition of the time evolution of Eq. (27) with a linear-depth circuit [46]. Our parallel RUS execution technique further allows for the parallel execution of rotation gates $e^{-iT_{pq}(c_p^\dagger c_q + c_q^\dagger c_p)\Delta\tau}$ and $e^{-iV_{pq}n_p n_q \Delta\tau}$ that appear for each fSWAP layer. Another promising system is the effective Hamiltonian derived from the down-folding technique [68], the time evolution of which can also be simulated by the combination of the fSWAP layer and the parallel RUS execution technique.

- (iii) *Combining other QEC codes.* Our injection protocol can be applied to any Calderbank-Shor-Steane (CSS) code beyond the surface code, offering room for improvement in the choice of the QEC code. Successfully integrating recently proposed quantum low-density parity-check (QLDPC) code families into our framework could reduce resource requirements further. While this would require changes in the compilation, the basic strategy remains the same: parallel RUS execution with efficient suppression of the injection overhead.

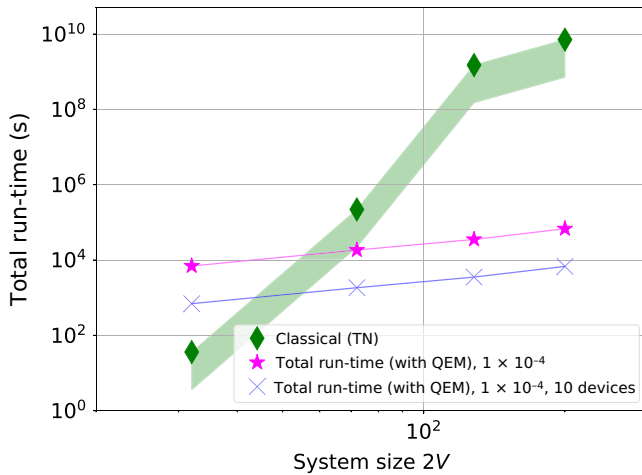


FIG. 19. A comparison of the total run-time for the QPE algorithm with that for classical tensor-network (TN) calculations. The horizontal axis indicates the system size $2V = 2M^2$ for the $M \times M$ Hubbard model. The green plot shows the run-time of the classical algorithm estimated in Ref. [15]. The blue plot displays the possible total run-time using ten independent quantum devices in our calculation.

We hope that our results will stimulate discussions about applications in the early FTQC era and accelerate the realization of practical quantum advantage.

Note added. We have recently received a private communication from the authors of Ref. [15]. They have claimed that, assuming $p = 10^{-4}$ in their estimation, the one-level distillation protocol becomes sufficient and the total number of physical qubits becomes less than 10^5 , which might lead to better or comparable performance than our framework. This is related to the observation that the qubitization approach significantly outperforms the Trotter approach for the quantum phase estimation of the 2D Hubbard model, as shown in Ref. [15]. However, if considering tasks where the Trotter approach performs comparably or better than qubitization approach, such as Hamiltonian simulations [60], our framework will offer substantially greater reductions in space-time resources compared to fully fledged FTQC. Further exploration is needed to understand when and how our approach provides advantage against the Clifford + T approach.

ACKNOWLEDGMENTS

K.F. is supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) Quantum Leap Flagship Program (MEXT Q-LEAP) through Grant No. JPMXS0120319794, by the Japan Science and Technology Agency (JST) COI-NEXT program through Grant No. JPMJPF2014, and by the JST Moonshot R&D program through Grant No. JPMJMS2061.

DATA AVAILABILITY

The numerical codes that support the findings of this article are available in Supplementary Material.

APPENDIX A: SURFACE CODE AND LATTICE SURGERY

In this section, we summarize the basics of the rotated surface code and lattice surgery used in this study.

1. Definition of the rotated surface code

We illustrate the rotated planar surface code in Fig. 20. This code is defined on physical qubits arranged in a $d \times d$ lattice (represented by white circles in Fig. 20). The stabilizer operators are defined on the faces of the lattice (shown as orange and blue faces in Fig. 20). The measurement circuit of the stabilizer operator and the order of the CNOT operation are provided in Fig. 21, with ancilla qubits used in the measurement circuit depicted as black dots in Fig. 20). The logical operators are defined on the lattice boundary, indicated by orange and blue solid lines.

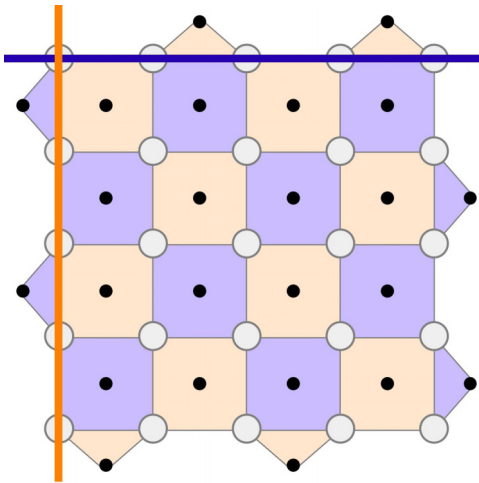


FIG. 20. The rotated surface code with code distance $d = 5$. The white and black circles represent physical qubits used for encoding and measurement, respectively. The X (Z) stabilizer operators are shown as orange (blue) faces. Representative logical X (Z) operators are given in orange (blue) solid lines on the boundary.

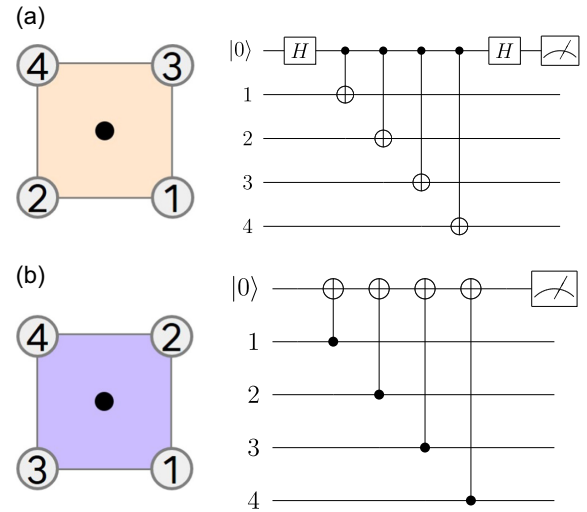


FIG. 21. Syndrome measurement circuits. (a) The X syndrome measurement circuit, with the order of CNOT operations represented by circled numbers on the left-hand side. (b) The Z syndrome measurement circuit.

The minimal length of the logical operator matches the side length of the lattice, making the code distance d .

2. Surface-code lattice surgery

The surface code inherently supports logical transversal CNOT gates owing to its CSS code structure. However, in practical scenarios in which entangling gates are limited to certain pairs of physical qubits, such as the 2D nearest-neighbor connectivity in superconducting devices, transversal CNOT gates are not feasible. The clever way to perform logical fault-tolerant gates in such a restricted scenario is known as lattice surgery [10,32]. Lattice surgery requires only the measurement of local stabilizer operators, which can easily be achieved even with restricted connectivity.

The fundamental lattice-surgery operations include patch merging, patch splitting, and patch deformation. By combining these fundamental operations with additional specific techniques for certain logical gates, we can perform all logical Clifford gates fault tolerantly. Below, we introduce how to perform H , CNOT, and S gates in this study.

a. Hadamard gate

For the rotated surface code, the logical Hadamard gate can be performed transversally. However, applying the transversal Hadamard gate rotates the patch surface by 90° [Fig. 22(a)]. To change the patch direction to its original one, we first deform the patch into a rectangular shape with a modified boundary [Fig. 22(b)]. Then, we adjust the boundary and shrink the rectangular patch into a square shape [Figs. 22(c)–22(e)]. Although this procedure brings

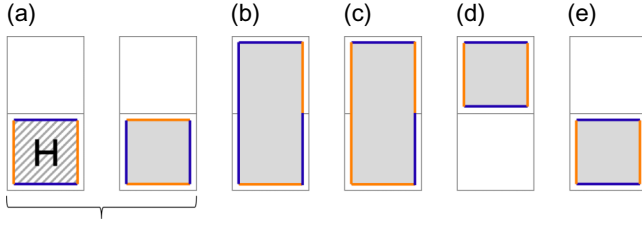


FIG. 22. A logical Hadamard gate using lattice surgery.

the patch almost back to its original position, it differs slightly by two lines of physical qubits (not shown in Fig. 22). To remedy this small shift, we use the “patch-sliding” technique [69]. The total execution time for this operation is $3d + 2$ code cycles = 3 clocks (In the following, we ignore the small constant-time operations in the count of the lattice-surgery clock). In certain cases, we do not need the patch to return to the initial position. Thus, the run-time decreases to 2 clocks.

b. CNOT and CZ gates

The CNOT gate is decomposed into sequential Pauli measurements using an ancilla state initialized to $|+\rangle$ as shown in Fig. 23. The $P \otimes P$ ($P = Z, X$) measurements can be performed by the combination of patch merging and patch splitting. The standard lattice-surgery operations are schematically shown in Fig. 24. It takes four patches and the total execution time is $3d$ code cycles = 3 clocks (note that if we can ignore the final moving operation, it finishes with $2d$ code cycles = 2 clocks). In Fig. 12, we use more compact implementation, as shown in Fig. 25, which only takes three patches. This compact implementation uses the symmetry among the ancilla and control qubits just after the first ZZ measurement.

The CZ gate is constructed using both the CNOT gate and the Hadamard gate. The CZ gate circuit and its lattice-surgery implementation are depicted in Figs. 26 and 27. The total execution time is 4 clocks.

c. Multi-target CNOT and multi-target CZ gates

In this study, we also use the multi-target CNOT and CZ gates to realize the controlled time-evolution operator. Using lattice surgery, multi-target CNOT gates can be executed without decomposing them into multiple

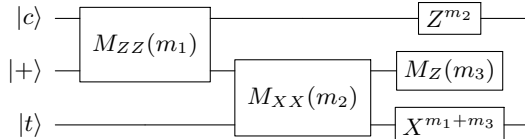


FIG. 23. A logical CNOT gate circuit using an ancilla qubit and $P \otimes P$ ($P = X, Z$) measurements. $|c\rangle$ and $|t\rangle$ indicate control and target qubits in the logical CNOT operation. The m_i ($i = 1, 2, 3$) are measured parities.

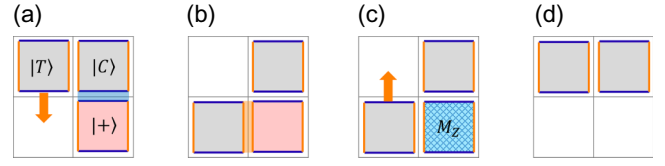


FIG. 24. Standard logical CNOT gate implementations using lattice surgery. (a) An ancilla state is initialized to $|+\rangle$ and then the first ZZ measurement is performed. (b) XX measurement between the ancilla state and $|T\rangle$. (c) The destructive logical Z measurement is performed on the ancilla state. (d) Moving $|T\rangle$ to the initial position.

single-target CNOT gates [58]. For a detailed example, see Fig. 28, where the multi-target CNOT operation takes a total of 8 clocks. Our compilation applies the multi-target CNOT only to a certain spin component (e.g. spin-down), as shown in Eq. (D6). By adopting the patch arrangement discussed in Sec. III F, we can reduce the run-time to 5 clocks, as shown in Fig. 29. This reduction assumes an initial division of spin-up and spin-down components into upper and lower rows, which introduces an additional patch-moving overhead of 3 clocks at the beginning and end of the entire Trotter time evolution. This differs slightly from our discussion in Sec. III F (see Fig. 7). Similarly to the cancellation of the multi-target CNOT gate, these patch-moving operations also cancel with each other between the Trotter steps.

The multi-target CZ gate operates similarly but includes additional H operations on target patches. Fortunately, the H operation is immediate (counted as zero clocks) and automatically rotates the patch, eliminating the need for the operations shown in Figs. 28(b) and 28(f), and the run-time becomes just 2 clocks. In the main text, since we need to perform multi-target CZ operations on both spin-up and spin-down components, the total run-time is $2 \times 2 = 4$ clocks.

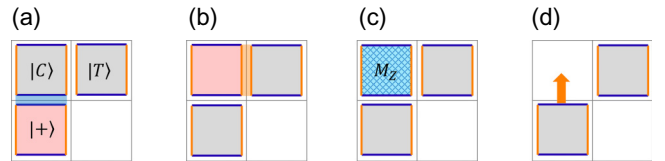


FIG. 25. More compact logical CNOT gate implementations. This implementation only uses three patches. (a) An ancilla state is initialized to $|+\rangle$ and then the first ZZ measurement is performed. Instead of moving the logical patch explicitly, we use the symmetry among the control and ancilla states to exchange them. (b) XX measurement between the ancilla state and $|T\rangle$. (c) The destructive logical Z measurement is performed on the ancilla state. (d) Moving $|T\rangle$ to the initial position.

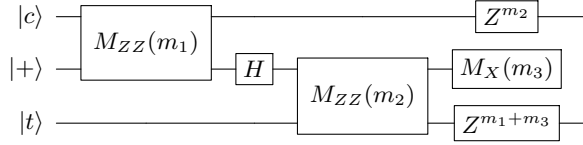


FIG. 26. A logical CZ gate circuit. This circuit can be obtained by combining the logical CNOT gate circuit shown in Fig. 23 and the logical Hadamard gate. Note that $|c\rangle$ and $|t\rangle$ are interchangeable. The m_i ($i = 1, 2, 3$) are measured parities.

d. S gate

The implementation of the S gate on the rotated planar surface code is notably complex since the surface code is not self-dual. Traditionally, the S gate has been realized through gate teleportation using a special resource state (the eigenstate of the Pauli Y operator) [9] or by performing a logical Y basis measurement using a rectangular-shaped patch [32]. Recently, a new method has been proposed for the fault-tolerant Y basis measurement within a single rotated planar surface-code patch [70]. This Y basis measurement enables us to perform an S gate with minimal space overhead (only two patches) as shown in the quantum circuit in Fig. 30. We employ this efficient method for the S gate implementation. The total execution time is $d + d/2$ code cycles = 1.5 clocks, as schematically shown in Fig. 31.

APPENDIX B: SMALL-ANGLE ANALOG ROTATION GATES

In this appendix, we discuss the implementation of the analog rotation gate. We first introduce the RUS protocol and then discuss the ancilla-state injection protocol [35], which is suitable for the small-angle rotation gates.

Instead of using Clifford + T decomposition, the STAR architecture directly performs the analog rotation gate by consuming a resource state, $|m_\theta\rangle_L \equiv R_{Z_L}(\theta)|+\rangle_L = \frac{1}{\sqrt{2}}(e^{-i\theta}|0\rangle_L + e^{+i\theta}|1\rangle_L)$, where the angle θ is assumed to be small (in this appendix, we show subscription L explicitly for logical states and operations). As shown

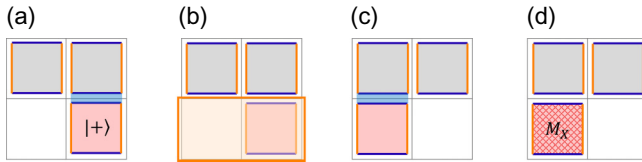


FIG. 27. A logical CZ gate using lattice surgery. (a) An ancilla state is initialized to $|+\rangle$ and then the first ZZ measurement is performed. (b) After the ZZ measurement, the transversal H gate is applied on the ancilla state. Using the orange region, the patch rotation as well as patch moving are performed. (c) The second ZZ measurement is performed. (d) The destructive logical X measurement on the ancilla patch is performed.

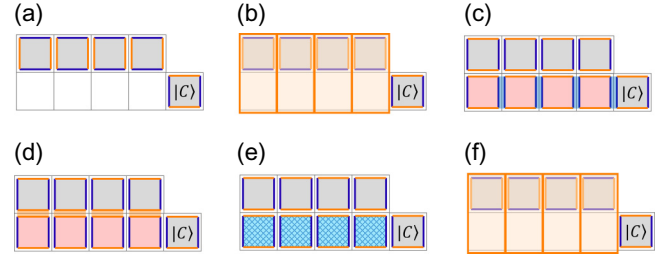


FIG. 28. A multi-target logical CNOT operation using lattice surgery. (a) The initial configuration with logical patches arranged as Fig. 6. We pick up the subset of the logical qubits shown in Fig. 6 for illustration. The lower-right patch is the control logical qubit and the other four patches are the CNOT operation targets. (b) First, we rotate the target patches. (c) Then, we prepare logical $|+\rangle$ in the ancilla regions (red patches) and perform logical ZZ measurements. These measurement results enable us to reproduce the parity between the control qubit and each ancilla patch. (d) After the ZZ measurements, we perform XX measurements between the ancilla patches and the target patches. (e) Then, the ancilla patches are measured on a Z basis. Using the measurement results, appropriate Pauli corrections are applied (in practice, appropriate Pauli corrections can be traced by software.) (f) The target patches are rotated again. In total, these sequences take 8 clocks.

in Fig. 1, the circuit for the analog rotation gate produces different output states depending on the measurement result in the $Z_L Z_L$ basis. If the measured value is $+1$ (-1), the output state is $R_{Z_L}(\theta)|\psi\rangle_L$ ($R_{Z_L}(-\theta)|\psi\rangle_L$). If we obtain $R_{Z_L}(-\theta)|\psi\rangle_L$, we perform the same circuit with updated angle $\theta' = 2\theta$ to compensate for the wrong rotation angle $-\theta$. The second circuit run is also probabilistic, so we repeat this procedure until obtaining the desired state $R_{Z_L}(\theta)|\psi\rangle_L$. The protocol is called the RUS protocol [16,31,47]. Each run of the circuit in Fig. 1 during the RUS protocol is called an “RUS trial,” and an instance of the protocol (a single rotation gate) is called an “RUS process.” On average, a single RUS process is completed in two trials.

Although all of the logical measurements in Fig. 1 are fault tolerant via the lattice surgery, the injection of the ancilla state $|m_\theta\rangle_L$ cannot be fault tolerant owing to the

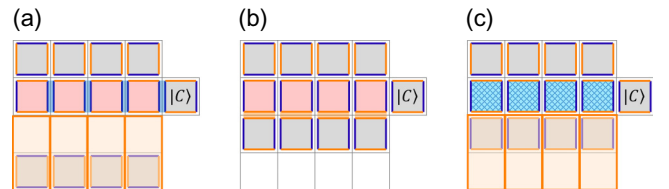


FIG. 29. Run-time reduction in the multi-target CNOT gate using the available ancilla region. (a) Patch rotations and ZZ measurements can be performed at the same time. (b) XX measurements. (c) Destructive Z measurements and patch rotations. The run-time decreases to 5 clocks in total.

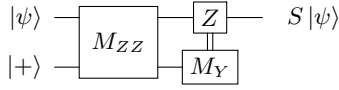


FIG. 30. A logical S gate circuit using the in-place Y measurement. M_{ZZ} is performed using the standard merging-splitting operation. The destructive Y measurement M_Y is performed by the technique proposed in Ref. [70]. Since M_{ZZ} and M_Y take d and $d/2$ code cycles for each operation, this circuit takes $d + d/2$ code cycles = 1.5 clocks in total.

analog nature of rotation angle. Therefore, the precision of the injected ancilla state directly affects the precision of the rotation gate. Here, we briefly introduce how to prepare the high-quality ancilla state for small-angle rotation [35]. The injection protocol is depicted in Fig. 32.

Let us consider the preparation of $|m_{\theta^*}\rangle_L$, where θ^* is a small target angle. First, we prepare a logical patch initialized in the $|+\rangle_L$ state. Then, we apply multiple rotation gates defined as

$$\prod_{i=0}^{k-1} R_{Z_{Q_i}}(\theta) = \prod_{i=0}^{k-1} [\cos \theta I_{Q_i} + i \sin \theta Z_{Q_i}], \quad (\text{B1})$$

where Q_i ($i = 0, \dots, k-1$) is i th subset of physical qubits, which satisfies $\bigcup_i Q_i = Q_{Z_L}$ and $|Q_i| > 1$, Q_{Z_L} is a subset of physical qubits that support the logical Z_L operator such that $\prod_{j \in Q_{Z_L}} Z_j = Z_L$. I_{Q_i} and Z_{Q_i} denote the identity and multi-qubit Z operator acting on the subset Q_i , respectively. k is a tunable parameter that determines how to divide Q_{Z_L} into Q_i s. An example of $k = 2$ is shown in Fig. 32(a). The resultant state after some Clifford corrections is a superposition of the target ancilla state and other states as follows:

$$\prod_{i=0}^{k-1} R_{Z_{Q_i}}(\theta) |+\rangle_L = \sqrt{p_{\text{ideal}}} |m_{\theta^*}\rangle_L + |\chi\rangle_L, \quad (\text{B2})$$

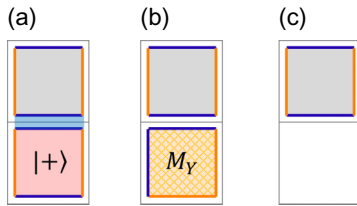


FIG. 31. Lattice-surgery operations in a logical S gate (the circuit of Fig. 30). Only one additional patch is required. (a) The logical ZZ measurement between the target patch and the ancilla patch is performed. (b) The in-place Y basis measurement on the ancilla patch is performed. This operation can be performed within $\lfloor d/2 \rfloor$ clocks [70]. (c) The final configuration.

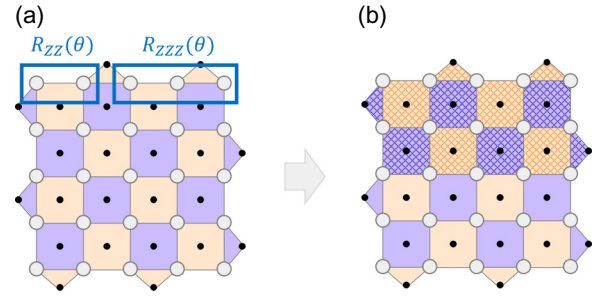


FIG. 32. The ancilla-state injection protocol in the STAR architecture. (a) The ancilla state is initialized in $|+\rangle_L$ first and then multiple rotation gates are acted along the logical Z_L operator. Multi- Z rotation gates can be performed using CNOT gates and Z rotation gates. (b) After acting the rotation gates, we perform a postselection on the hatched stabilizers to remove critical errors that bring leading logical errors. Errors not detected in the postselection can be corrected in the subsequent quantum error-correction (QEC) procedure safely.

where p_{ideal} is defined as

$$p_{\text{ideal}}(\theta, k) = \sin^{2k} \theta + \cos^{2k} \theta \quad (\text{B3})$$

and $|\chi\rangle_L$ is the superposition of the logical $|+\rangle$ states on which some Pauli Z operators act,

$$|\chi\rangle = \sum_{(n_1, \dots, n_k) \neq (0, \dots, 0), (1, \dots, 1)} c_{n_1, \dots, n_k} \prod_j Z_{Q_j}^{n_j} |+\rangle_L. \quad (\text{B4})$$

The angle θ is related to θ^* as follows:

$$\theta^* = \sin^{-1} \left(\frac{1}{\sqrt{p_{\text{ideal}}}} \sin^k(\theta) \right) \approx \theta^k + \mathcal{O}(\theta^{k+2}). \quad (\text{B5})$$

States in $|\chi\rangle_L$ belong to a Hilbert subspace with non-trivial syndrome values owing to additional Z operators acting on these states. Therefore, we can obtain the target ancilla state by the postselection of the syndrome measurement result. The p_{ideal} value denotes the probability of obtaining the target ancilla state in case of no quantum noise and is nearly identity when the rotation angle is very small, $p_{\text{ideal}}(\theta, k) \approx 1 - k\theta^2 + \mathcal{O}(\theta^4)$. In the noisy case, the quantum noise on $|\chi\rangle_L$ forms undetectable logical errors. The worst-case logical error probability including the effect of the RUS protocol is given as Eq. (1).

In practice, the postselection region can be optimized to maximize its success probability by combining QEC. The stabilizer operators checked in the postselection are shown by the hatched pattern in Fig. 32(b). Errors outside the postselected region can be corrected by subsequent QEC procedures and do not lead to leading-order logical errors. Our state injection protocol, which can be performed within a single patch, eliminates the need for an “ancilla-state factory.” Due to the compactness of our rotation gate, we can perform multiple rotation gates in parallel

without additional space overhead provided that there are enough free patches for the state injection. This “easy-to-parallelize” feature of the STAR architecture contrasts significantly with existing FTQC architectures, where parallel T gates require many more physical qubits to parallelize the magic state distillation.

APPENDIX C: DERIVATION OF EQ. (16)

In this appendix, we outline the derivation of Eq. (16). Let us consider the case that involves performing M independent rotation gates in parallel. To estimate the average RUS trial number needed to complete all M processes, we first calculate the probability that the parallel RUS execution finishes at the K th RUS trial, P_K^M . This scenario can be divided further, into M independent cases: $M - 1$ processes finish before the K th trial and then the remaining process finishes on the K th trial; $M - 2$ processes finish before the K th trial, and the remaining two processes finish on the K th trial; and so on. Since the single RUS trial finishes with a probability of $1/2$, we can calculate the probability of these M cases. For example, the probability of the first case is the following:

$$\begin{aligned} P_{\text{one process finishes at } K}^M &= M \left(\sum_{n=1}^{K-1} \left(\frac{1}{2} \right)^n \right)^{M-1} \times \left(\frac{1}{2} \right)^K \\ &= MA^{M-1}B, \end{aligned} \quad (\text{C1})$$

where we define $A \equiv \sum_{n=1}^{K-1} (1/2)^n$ and $B \equiv (1/2)^K$. Factor M represents the number of choices for which process finishes on the K th trial. A^{M-1} is a sum of probability for all possible patterns that $M - 1$ RUS processes finish before the K th RUS trial. Other cases are calculated similarly. As a result, the sum of probabilities of all cases, which is P_K^M , is given as

$$\begin{aligned} P_K^M &= MA^{M-1}B + {}_MC_2 A^{M-2}B^2 + \dots + B^M \\ &= (A + B)^M - A^M, \end{aligned} \quad (\text{C2})$$

where the ${}_MC_L$ are the binomial coefficients, and we use the binomial expansion $(a + b)^M = a^M + Ma^{M-1}b + \dots + b^M$ at the second equality. Therefore, the average number of RUS trials needed to complete all M RUS processes is provided as follows:

$$\begin{aligned} \langle K \rangle_M &= \sum_K KP_K^M \\ &= \sum_K K \left[(1 - 2^{-K})^M - (1 - 2^{-K+1})^M \right], \end{aligned} \quad (\text{C3})$$

where we use $\sum_{n=1}^L (1/2)^n = 1 - 2^{-L}$.

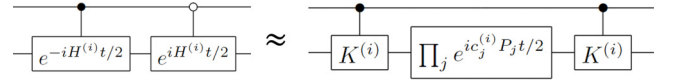


FIG. 33. An example of the controlled time evolution by a combination of the controlled multi-qubit Pauli operator and control-free time evolution.

APPENDIX D: CONTROLLED TIME-EVOLUTION OPERATOR

The controlled time-evolution operator appears in several quantum algorithms. A typical example is the Hadamard test circuit shown in Fig. 17. The naive approach to implementing controlled time evolution involves replacing each rotation gate $e^{-ic_j P_j \tau}$ in the Trotter step with the controlled counterparts. However, this naive replacement introduces an overhead proportional to the number of interaction terms, which becomes impractical for large system size. Fortunately, for certain Hamiltonians, efficient implementation of controlled time evolution is possible with a reasonable overhead [71]. Let us consider a Hamiltonian consisting of a certain set of sub-Hamiltonians,

$$H = \sum_l H^{(l)}, \quad H^{(l)} = \sum_j c_j^{(l)} P_j, \quad (\text{D1})$$

where P_j is a multi-qubit Pauli operator and $c_j^{(l)}$ is a coefficient of the sub-Hamiltonian $H^{(l)}$. Let $K^{(l)}$ be a certain multi-qubit Pauli operator that anticommutes with the sub-Hamiltonian $H^{(l)}$, $K^{(l)} H^{(l)} K^{(l)} = -H^{(l)}$. This relationship leads to $K^{(l)} e^{+iH^{(l)}t} K^{(l)} = e^{-iH^{(l)}t}$. In this case, we can easily construct a controlled time-evolution operator by combining the control-free time evolution and controlled- $K^{(l)}$ operations, as shown in Fig. 33. This method is well suited for our purpose, since it allows us to leverage the compilation results directly.

For the 2D Hubbard model, we can choose the sub-Hamiltonian as follows:

$$H = H^{(0)} + H^{(1)}, \quad (\text{D2})$$

$$H^{(0)} = -\frac{t}{2} \sum_{\langle i,j \rangle, \sigma} (X_{i,\sigma} X_{j,\sigma} + Y_{i,\sigma} Y_{j,\sigma}) Z_{ij,\sigma}^{\leftrightarrow}, \quad (\text{D3})$$

$$H^{(1)} = \frac{U}{4} \sum_i Z_{i,\uparrow} Z_{i,\downarrow}. \quad (\text{D4})$$

In this choice, $K^{(l)}$ ($l = 0, 1$) can be chosen as

$$K^{(0)} = \prod_{k \in V^{(0)}, \sigma} Z_{k,\sigma}, \quad (\text{D5})$$

$$K^{(1)} = \prod_{k \in V^{(1)}} X_{k,\downarrow}, \quad (\text{D6})$$

where the $V^{(l)}$ ($l = 0, 1$) are defined as

$$V^{(0)} = \{k = (x, y) \mid x + y = 1 \pmod{2}\}, \quad (\text{D7})$$

$$V^{(1)} = \{\forall k = (x, y)\}, \quad (\text{D8})$$

where we introduce the 2D coordinate of the Hubbard-model lattice for the first time, $k = (x, y)$ ($x = 0, \dots, N - 1, y = 0, \dots, N - 1$). We show the example of $V^{(0)}$ on the 4×4 lattice in Fig. 34. Furthermore, since the control qubit is common for all CNOT and CZ operations, multiple operations can be combined into a single multi-target operation. For example, we can merge multiple CNOT gates with the same control qubit into a single multi-target CNOT gate using an appropriate lattice-surgery operation [58] (for more details, see Appendix A 2 c). The multi-target CNOT gate has a constant time overhead, allowing us to minimize additional time overhead for controlled time evolution. Typically, four multi-target CNOT layers and four multi-target CZ layers are required for each Trotter step to construct the controlled time-evolution operator. However, two multi-target CZ layers in the middle of the Trotter formula cancel each other out. In addition, the first and last multi-target CNOT layers cancel with those from the previous and next Trotter steps. By considering the lattice-surgery clock discussed in Appendix A 2 c, the total overhead of the controlled time-evolution operator, consisting of T Trotter steps, can be estimated as follows:

$$T_{\text{ctrl}} = 2 \times 5 + 2 \times 3 + T(2 \times 5 + 2 \times 4) = 16 + 18T, \quad (\text{D9})$$

where the first two terms correspond to the overhead of multi-target CNOT and patch-moving operations at the beginning and the end of the Trotter time evolution.

APPENDIX E: QCELS ALGORITHM

In the following, we first introduce the main idea of the QCELS algorithm in a simple setup and then cite important results from Ref. [21] as Algorithm 1 and Theorem E1, which we need for resource estimation.

The QCELS algorithm relies on a very simple idea to extract the eigenvalues. Let us consider that we have the time-series data of the expectation value, $Z_n = \langle \psi | e^{-it_n H} | \psi \rangle$ ($t_n = n\tau$ for $n = 0, 1, \dots, N - 1$). If the initial state $|\psi\rangle$ overlaps the ground state $|E_0\rangle$ largely, $p_0 \approx 1$, the expectation value approximately behaves as

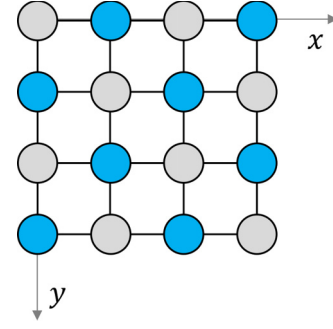


FIG. 34. The targets of the CZ operations on the 2D lattice (4×4). The blue nodes are the target of the CZ operations. Here, we only show spin-up components. The upper-left node is the origin of the coordinates, $(x, y) = (0, 0)$.

$Z_n \approx p_0 e^{-iE_0 t_n}$. Therefore we can extract the energy eigenvalue by solving the following nonlinear least-squares problem:

$$(r^*, \theta^*) = \underset{r, \theta}{\operatorname{argmin}} L(r, \theta), \quad (\text{E1})$$

$$L(r, \theta) = \frac{1}{N} \sum_{n=0}^{N-1} |Z_n - r e^{-it_n \theta}|^2. \quad (\text{E2})$$

This optimization problem can be solved classically. The optimal θ^* provides an approximation of the energy eigenvalue E_0 . This optimization subroutine is why the algorithm is named “quantum complex exponential least squares” (QCELS).

The basic concept of the QCELS algorithm is straightforward but to achieve Heisenberg-limited scaling, the actual implementation is more complex. The whole algorithm, called “multi-level QCELS,” is cited in Algorithm 1 to make this paper self-contained. In multi-level QCELS, we start with a small time-evolution interval τ_1 and estimate the eigenvalue using the time-series data $Z_n^1 = \langle \psi | e^{-in\tau_1 H} | \psi \rangle$ ($n = 0, 1, \dots, N - 1$). Small τ_1 results in a rough estimate of E_0 ; thus, we refine this estimation by increasing the time interval τ_1 to $\tau_j = 2^{j-1} \tau_1$ ($j = 1, \dots, J$), where $J \propto \log_2(\delta/N\epsilon)$, δ is a small constant (their definitions are provided in Theorem E1 cited below), and ϵ is the target precision of the estimation of E_0 . Performing the estimation of E_0 J times with increasing τ_j , we achieve an estimation of E_0 with precision ϵ . The computational complexity of Algorithm 1 is stated in Theorem E1, which introduces key parameters for resource estimation.

Theorem E1 (Ref. [21], p. 9): Let θ^* be the output of Algorithm 1. Given $p_0 > 0.71$, $0 < \eta < 1/2$, $0 < \epsilon <$

ALGORITHM 1. Multi-level QCELS algorithm in Ref. [21], p. 8.

-
-
- 1: **Preparation:** Number of data pairs: N ; number of samples: N_s ; number of iterations: J ; sequence of time steps: $\{\tau_j\}_{j=1}^J$; Quantum oracle: $\{\exp(-i\tau_j H)\}_{j=1}^J$;
 - 2: **Running:**
 - 3: $\lambda_{\min} \leftarrow -\pi$; $\lambda_{\max} \leftarrow \pi$; $\triangleright [\lambda_{\min}, \lambda_{\max}]$ contains λ_0
 - 4: $j \leftarrow 1$;
 - 5: **for** $j = 1, \dots, J$ **do**
 - 6: Generate the data set in $\mathcal{D}_H = \{(n\tau_j, Z_n)\}_{n=0}^{N-1}$ using the circuit in Fig. 17 with $t_n = n\tau_j$.
 - 7: Define loss function $L(r, \theta)$ according to Eq.(E2).
 - 8: Minimizing the loss function.

$$(r_j^*, \theta_j^*) \leftarrow \underset{r \in \mathbb{C}, \theta \in [-\lambda_{\min}, \lambda_{\max}]}{\operatorname{argmin}} L(r, \theta),$$

- 9: $\lambda_{\min} \leftarrow \theta_j^* - \frac{\pi}{2\tau_j}$; $\lambda_{\max} \leftarrow \theta_j^* + \frac{\pi}{2\tau_j}$ $\triangleright$ Shrink the search interval by $1/2$
 - 10: **end for**
 - 11: **Output:** θ^*
-
-

$1/2$, we can choose δ according to $\delta = \Theta(\sqrt{1 - p_0})$,

$$\begin{aligned} J &= \lceil \log_2(1/\epsilon) \rceil + 1, \\ \tau_j &= 2^{j-1 - \lceil \log_2(1/\epsilon) \rceil} \frac{\delta}{N\epsilon}, \quad \forall 1 \leq j \leq J. \end{aligned} \quad (\text{E3})$$

Choose $NN_s = \tilde{\Theta}(\delta^{-(2+o(1))})$. Then,

$$T_{\max} = N\tau_J = \frac{\delta}{\epsilon}, \quad (\text{E4})$$

$$T_{\text{total}} = \sum_{j=1}^J N(N-1)N_s\tau_j/2 = \tilde{\Theta}(\delta^{-(1+o(1))}\epsilon^{-1}) \quad (\text{E5})$$

and

$$\mathbb{P}(|(\theta^* - \lambda_0) \bmod [-\pi, \pi]| < \epsilon) \geq 1 - \eta.$$

The resource estimation in this study relies on the complexity of Theorem E1 cited above.

APPENDIX F: RUN-TIME OF THE SECOND-ORDER TROTTER DECOMPOSITION USING THE SERIAL-COMPILATION STRATEGY

In this appendix, we derive the run-time for the second-order Trotter decomposition using the serial-compilation strategy [32]. To begin with, we count the number of rotations required and how to perform them. We consider the same Hamiltonian as in the main text, namely Eq. (11). In this Hamiltonian, there are V ZZ rotation terms, $4(V-M)$ XZ...ZX rotation terms, and $4(V-M)$ YZ...ZY rotation terms. The second-order Trotter decomposition

doubles these numbers except for the two rotation layers in the middle, which can be merged into a single rotation layer. For the purpose of this estimation, we place the ZZ rotation gate layer in the middle of the decomposition. Therefore, the run-time of the second-order Trotter decomposition using the serial compilation is given as

$$VT_{ZZ} + 2 \times 4(V-M)T_{XZ\dots ZX} + 2 \times 4(V-M)T_{YZ\dots ZY}, \quad (\text{F1})$$

where T_P is an average run-time of the Pauli P rotation gate.

Next, we discuss the run-time estimation of each rotation gate. In the context of estimating the RUS overhead, we assume that the injection overhead can be hidden by separating the injection patches from the calculation block, as discussed in Ref. [31]. Moreover, since each rotation gate is performed sequentially, the average trial of the RUS protocol is 2. Therefore, assuming that Pauli measurements require 1 clock, the average run-time of each RUS protocol is 2 clocks. For the ZZ rotation gates, no basis change operations are needed, allowing them to be performed in $T_{ZZ} = 2$ clocks. The XZ...ZX rotation gates need patch rotation at the beginning and the end, resulting in an average run-time of $T_{XZ\dots ZX} = 3 \times 2 + 2 = 8$ clocks. Similarly, the YZ...ZY rotation gates need patch rotation, and the S gates at the beginning and the end, leading to an average run-time of $T_{YZ\dots ZY} = 3 \times 2 + 1.5 \times 2 + 2 = 11$ clocks on average. Therefore, by inserting these values in Eq. (F1), we obtain the following:

$$\begin{aligned} &V \times 2 + 2 \times 4(V-M) \times 8 + 2 \times 4(V-M) \times 11 \\ &= 154V - 152M. \end{aligned} \quad (\text{F2})$$

The comparison between the serial- and parallel-compilation results is displayed in Fig. 16.

-
- [1] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Rev.* **41**, 303 (1999).
 - [2] D. S. Abrams and S. Lloyd, Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors, *Phys. Rev. Lett.* **83**, 5162 (1999).
 - [3] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, Simulated quantum computation of molecular energies, *Science* **309**, 1704 (2005).
 - [4] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum algorithm for linear systems of equations, *Phys. Rev. Lett.* **103**, 150502 (2009).
 - [5] J. Preskill, Quantum computing in the NISQ era and beyond, *Quantum* **2**, 79 (2018).
 - [6] A. Kitaev, Fault-tolerant quantum computation by anyons, *Ann. Phys.* **303**, 2 (2003).
 - [7] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, *J. Math. Phys.* **43**, 4452 (2002).

- [8] R. Raussendorf and J. Harrington, Fault-tolerant quantum computation with high threshold in two dimensions, *Phys. Rev. Lett.* **98**, 190504 (2007).
- [9] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, *Phys. Rev. A* **86**, 032324 (2012).
- [10] C. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, Surface code quantum computing by lattice surgery, *New J. Phys.* **14**, 123011 (2012).
- [11] Y. Zhao, Y. Ye, H.-L. Huang, Y. Zhang, D. Wu, H. Guan, Q. Zhu, Z. Wei, T. He, S. Cao *et al.*, Realization of an error-correcting surface code with superconducting qubits, *Phys. Rev. Lett.* **129**, 030501 (2022).
- [12] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann *et al.*, Realizing repeated quantum error correction in a distance-three surface code, *Nature* **605**, 669 (2022).
- [13] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush *et al.*, Suppressing quantum errors by scaling a surface code logical qubit, *Nature* **614**, 676 (2023).
- [14] C. Gidney and M. Ekerå, How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, *Quantum* **5**, 433 (2021).
- [15] N. Yoshioka, T. Okubo, Y. Suzuki, Y. Koizumi, and W. Mizukami, Hunting for quantum-classical crossover in condensed matter problems, *npj Quantum Inf.* **10**, 45 (2024).
- [16] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, Elucidating reaction mechanisms on quantum computers, *Proc. Natl. Acad. Sci.* **114**, 7555 (2017).
- [17] J. J. Goings, A. White, J. Lee, C. S. Tautermann, M. Degroote, C. Gidney, T. Shiozaki, R. Babbush, and N. C. Rubin, Reliably assessing the electronic structure of cytochrome P450 on today's classical computers and tomorrow's quantum computers, *Proc. Natl. Acad. Sci.* **119**, e2203533119 (2022).
- [18] L. Lin and Y. Tong, Heisenberg-limited ground-state energy estimation for early fault-tolerant quantum computers, *PRX Quantum* **3**, 010318 (2022).
- [19] K. Wan, M. Berta, and E. T. Campbell, Randomized quantum algorithm for statistical phase estimation, *Phys. Rev. Lett.* **129**, 030503 (2022).
- [20] R. Kshirsagar, A. Katabarwa, and P. D. Johnson, On proving the robustness of algorithms for early fault-tolerant quantum computers, *Quantum* **8**, 1531 (2024).
- [21] Z. Ding and L. Lin, Even shorter quantum circuit for phase estimation on early fault-tolerant quantum computers with applications to ground-state energy estimation, *PRX Quantum* **4**, 020331 (2023).
- [22] Y. Suzuki, S. Endo, K. Fujii, and Y. Tokunaga, Quantum error mitigation as a universal error reduction technique: Applications from the NISQ to the fault-tolerant quantum computing eras, *PRX Quantum* **3**, 010345 (2022).
- [23] C. Piveteau, D. Sutter, S. Bravyi, J. M. Gambetta, and K. Temme, Error mitigation for universal gates on encoded qubits, *Phys. Rev. Lett.* **127**, 200505 (2021).
- [24] N. P. Breuckmann and J. N. Eberhardt, Quantum low-density parity-check codes, *PRX Quantum* **2**, 040101 (2021).
- [25] P. Panteleev and G. Kalachev, Degenerate quantum LDPC codes with good finite length performance, *Quantum* **5**, 585 (2021).
- [26] C. Gidney, M. Newman, P. Brooks, and C. Jones, Yoked surface codes, *Nat. Commun.* **16**, 4498 (2025).
- [27] Y. Hong, M. Marinelli, A. M. Kaufman, and A. Lucas, Long-range-enhanced surface codes, *Phys. Rev. A* **110** (2024).
- [28] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, *Nature* **627**, 778 (2024).
- [29] H. Goto, High-performance fault-tolerant quantum computing with many-hypercube codes, *Sci. Adv.* **10** (2024).
- [30] S. Yoshida, S. Tamiya, and H. Yamasaki, Concatenate codes, save qubits, *npj Quantum Inf.* **11** (2025).
- [31] Y. Akahoshi, K. Maruyama, H. Oshima, S. Sato, and K. Fujii, Partially fault-tolerant quantum computing architecture with error-corrected Clifford gates and space-time efficient analog rotations, *PRX Quantum* **5**, 010337 (2024).
- [32] D. Litinski, A game of surface codes: Large-scale quantum computing with lattice surgery, *Quantum* **3**, 128 (2019).
- [33] S. Bravyi and J. Haah, Magic-state distillation with low overhead, *Phys. Rev. A* **86**, 052329 (2012).
- [34] N. J. Ross and P. Selinger, Optimal ancilla-free Clifford + T approximation of z -rotations, *Quantum Inf. Comput.* **16**, 901 (2016).
- [35] R. Toshio, Y. Akahoshi, J. Fujisaki, H. Oshima, S. Sato, and K. Fujii, Practical quantum advantage on partially fault-tolerant quantum computer, *Phys. Rev. X* **15**, 021057 (2025).
- [36] H. Choi, F. T. Chong, D. Englund, and Y. Ding, Fault tolerant non-Clifford state preparation for arbitrary rotations, *arXiv:2303.17380* [quant-ph].
- [37] D. Herr, F. Nori, and S. J. Devitt, Optimization of lattice surgery is NP-hard, *npj Quantum Inf.* **3**, 35 (2017).
- [38] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. G. Almudever, Mapping of lattice surgery-based quantum circuits on surface code architectures, *Quantum Sci. Technol.* **4**, 015005 (2018).
- [39] J. Lee, Y. Kang, J. Ha, and J. Heo, Lattice surgery-based surface code architecture using remote logical CNOT operation, *Quantum Inf. Process.* **21**, 217 (2022).
- [40] M. Beverland, V. Kliuchnikov, and E. Schoute, Surface code compilation via edge-disjoint paths, *PRX Quantum* **3**, 020342 (2022).
- [41] K. Hamada, Y. Suzuki, and Y. Tokunaga, Efficient and high-performance routing of lattice-surgery paths on three-dimensional lattice, *arXiv:2401.15829* [quant-ph].
- [42] D. B. Tan, M. Y. Niu, and C. Gidney, *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA) (IEEE, 2024)*, pp. 325-339.
- [43] J. Hubbard and B. H. Flowers, Electron correlations in narrow energy bands, *Proc. R. Soc. Lond. A: Math. Phys. Sci.* **276**, 238 (1963).
- [44] E. T. Campbell, Early fault-tolerant simulations of the Hubbard model, *Quantum Sci. Technol.* **7**, 015007 (2021).
- [45] I. D. Kivlichan, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, W. Sun, Z. Jiang, N. Rubin, A. Fowler, A.

- Aspuru-Guzik, H. Neven, and R. Babbush, Improved fault-tolerant quantum simulation of condensed-phase correlated electrons via Trotterization, *Quantum* **4**, 296 (2020).
- [46] I. D. Kivlichan, J. McClean, N. Wiebe, C. Gidney, A. Aspuru-Guzik, G. K.-L. Chan, and R. Babbush, Quantum simulation of electronic structure with linear depth and connectivity, *Phys. Rev. Lett.* **120**, 110501 (2018).
- [47] N. C. Jones, J. D. Whitfield, P. L. McMahon, M.-H. Yung, R. V. Meter, A. Aspuru-Guzik, and Y. Yamamoto, Faster quantum chemistry simulation on fault-tolerant quantum computers, *New J. Phys.* **14**, 115023 (2012).
- [48] K. Temme, S. Bravyi, and J. M. Gambetta, Error mitigation for short-depth quantum circuits, *Phys. Rev. Lett.* **119**, 180509 (2017).
- [49] S. Endo, S. C. Benjamin, and Y. Li, Practical quantum error mitigation for near-future applications, *Phys. Rev. X* **8**, 031027 (2018).
- [50] E. Dagotto, Correlated electrons in high-temperature superconductors, *Rev. Mod. Phys.* **66**, 763 (1994).
- [51] D. P. Arovas, E. Berg, S. A. Kivelson, and S. Raghu, The Hubbard model, *Ann. Rev. Condens. Matter Phys.* **13**, 239 (2022).
- [52] R. Somma, G. Ortiz, J. E. Gubernatis, E. Knill, and R. Laflamme, Simulating physical phenomena by quantum networks, *Phys. Rev. A* **65**, 042323 (2002).
- [53] T. Hagge, Optimal fermionic swap networks for Hubbard models, [arXiv:2001.08324](https://arxiv.org/abs/2001.08324) [quant-ph].
- [54] Actually, if we consider the 32 parallel RUS execution runtime distribution under the assumption that the injection takes 1 clock (as discussed in Sec. III G) and use N_{total} in Table II, the deviation is just about 0.075% of the true average value in 99% confidence.
- [55] Note that this rearrangement is just a modification of the position of the logical patch, not a modification of the JW ordering. Therefore, this operation does not need the fSWAP operations.
- [56] C. Gidney, Stim: A fast stabilizer circuit simulator, *Quantum* **5**, 497 (2021).
- [57] See the Supplemental Material at <http://link.aps.org/supplemental/10.1103/93zr-lykb> for the numerical codes that generate Fig. 15 and for the numerical codes for resource estimation.
- [58] D. Litinski and F. v. Oppen, Lattice surgery with a twist: Simplifying Clifford gates of surface codes, *Quantum* **2**, 62 (2018).
- [59] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information* (Cambridge University Press, Cambridge, England, 2000).
- [60] A. M. Childs, D. Maslov, Y. Nam, N. J. Ross, and Y. Su, Toward the first quantum simulation with quantum speedup, *Proc. Natl. Acad. Sci.* **115**, 9456 (2018).
- [61] E. Campbell, Random compiler for fast Hamiltonian simulation, *Phys. Rev. Lett.* **123**, 070503 (2019).
- [62] Y. Ouyang, D. R. White, and E. T. Campbell, Compilation by stochastic Hamiltonian sparsification, *Quantum* **4**, 235 (2020).
- [63] T. Kosugi, Y. Nishiya, H. Nishi, and Y.-i. Matsushita, Imaginary-time evolution using forward and backward real-time evolution with a single ancilla: First-quantized eigensolver algorithm for quantum chemistry, *Phys. Rev. Res.* **4**, 033121 (2022).
- [64] K. Mizuta, Y. O. Nakagawa, K. Mitarai, and K. Fujii, Local variational quantum compilation of large-scale Hamiltonian dynamics, *PRX Quantum* **3**, 040302 (2022).
- [65] S. Kanasugi, S. Tsutsui, Y. O. Nakagawa, K. Maruyama, H. Oshima, and S. Sato, Computation of Green's function by local variational quantum compilation, *Phys. Rev. Res.* **5**, 033070 (2023).
- [66] K. Kanno, M. Kohda, R. Imai, S. Koh, K. Mitarai, W. Mizukami, and Y. O. Nakagawa, Quantum-selected configuration interaction: classical diagonalization of Hamiltonians in subspaces selected by quantum computers, [arXiv:2302.11320](https://arxiv.org/abs/2302.11320) [quant-ph].
- [67] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K.-L. Chan, Low-depth quantum simulation of materials, *Phys. Rev. X* **8**, 011044 (2018).
- [68] S. Kanno, S. Endo, T. Utsumi, and T. Tada, Resource estimations for the Hamiltonian simulation in correlated electron materials, *Phys. Rev. A* **106**, 012612 (2022).
- [69] M. McEwen, D. Bacon, and C. Gidney, Relaxing hardware requirements for surface code circuits using time-dynamics, *Quantum* **7**, 1172 (2023).
- [70] C. Gidney, Inplace access to the surface code Y basis, *Quantum* **8**, 1310 (2024).
- [71] Y. Dong, L. Lin, and Y. Tong, Ground-state preparation and energy estimation on early fault-tolerant quantum computers via quantum eigenvalue transformation of unitary matrices, *PRX Quantum* **3**, 040305 (2022).