Universal Quantum Computing with Twist-Free and Temporally Encoded Lattice Surgery

Christopher Chamberland[1,2,*] and Earl T. Campbell[3]

[1] *AWS Center for Quantum Computing, Pasadena, California 91125, USA*
[2] *IQIM, California Institute of Technology, Pasadena, California 91125, USA*
[3] *AWS Center for Quantum Computing, Cambridge, CB1 2GA, United Kingdom*

(Received 16 November 2021; accepted 18 January 2022; published 25 February 2022)

Lattice-surgery protocols allow for the efficient implementation of universal gate sets with two-dimensional topological codes where qubits are constrained to interact with one another locally. In this work, we first introduce a decoder capable of correcting spacelike and timelike errors during lattice-surgery protocols. Subsequently, we compute the logical failure rates of a lattice-surgery protocol for a biased circuit-level noise model. We then provide a protocol for performing twist-free lattice surgery, where we avoid twist defects in the bulk of the lattice. Our twist-free protocol eliminates the extra circuit components and gate-scheduling complexities associated with the measurement of higher weight stabilizers when using twist defects. We also provide a protocol for temporally encoded lattice surgery that can be used to reduce both the run times and the total space-time costs of quantum algorithms. Lastly, we propose a layout for a quantum processor that is more efficient for rectangular surface codes exploiting noise bias and that is compatible with the other techniques mentioned above.

DOI: 10.1103/PRXQuantum.3.010331

I. INTRODUCTION

Fault-tolerant quantum computing architectures enable the protection of logical qubits from errors by encoding them in error-correcting codes, while simultaneously allowing for gates to be performed on such qubits. Importantly, failures arising during the implementation of logical gates do not result in uncorrectable errors as long as the total number of such failures remains below a certain fraction of the code distance [1–5]. In most practical settings, quantum logic gates are split into two categories. The first category corresponds to Clifford operations, which can be efficiently simulated by classical computers. The second category corresponds to non-Clifford operations, which cannot be efficiently simulated using purely classical resources. Early proposals for fault-tolerant quantum computation used transversal gates to perform logical Clifford operations [6]. Later, it has been shown that by braiding defects in a surface code, some Clifford operations can be realized fault-tolerantly in a two-dimensional (2D) local architecture with a high threshold [7]. Recently, lattice surgery [8] has replaced the braiding approach due to its ability to retain locality constraints and high thresholds (features that are required by many hardware architectures), while additionally offering a much lower resource cost [9–12]. These approaches all perform non-Clifford gates by teleportation [13,14] of magic states prepared by some distillation procedure [15–21]. Alternative ideas have been proposed for circumventing the need for magic states [22–25] but detailed studies [26–28] have not found any of these alternatives to be competitive for a wide range of failure rates below the surface-code threshold.

Our work introduces the following key results. After briefly reviewing the model of Pauli-based computation and its implementation via lattice surgery in Sec. II, we then explicitly provide a decoder compatible with lattice surgery in Sec. III. In particular, our decoder is capable of correcting both spacelike and timelike errors that occur during lattice-surgery protocols. We then perform simulations of an $X \otimes X$ Pauli measurement using a biased circuit-level noise model.

In Sec. IV, we introduce a twist-free approach for measuring arbitrary Pauli operators using the surface code. Our approach avoids the extra circuit and gate-scheduling complexities that arise when using twists, where by twists we refer to lattices that contain twist defects in the bulk (see, e.g., Fig. 1 in Sec. II B). We show that the approximate cost of avoiding twists is a 2× slowdown in the algorithm

[*] mathematicschris@gmail.com

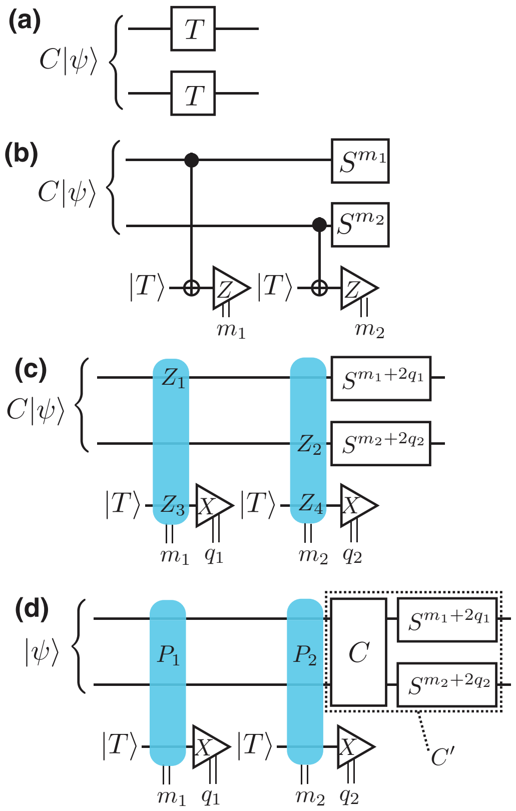

FIG. 1. Equivalent approaches to implementing two T gates. The blue rounded rectangles show multiqubit Pauli measurements. (a) A simple unitary circuit approach. (b) The use of gate teleportation with $|T\rangle := (|0\rangle + e^{i\pi/4}|1\rangle)/\sqrt{2}$ magic states, controlled-NOT (CNOT) gates, Pauli Z measurements with outcomes m_1 and m_2, and classical conditioned Cliffords based on these outcomes. The conditional S gates are $S = |0\rangle\langle 0| + i|1\rangle\langle 1|$. (c) The use of gate teleportation with the CNOT gates replaced by two-qubit Pauli measurements. (d) Given an input state $|\psi\rangle$ that carries a Clifford frame correction C, we conjugate C through the circuit so that the multiqubit Pauli measurements are now $P_1 = CZ_1Z_3C^\dagger$ and $P_2 = CZ_2Z_4C^\dagger$ (which commute) and the output state carries a new Clifford frame correction $C' = S_1^{m_1+2q_1}S_2^{m_2+2q_2}C$. This last circuit represents the standard PBC approach for computing $T^{\otimes 2}$.

run time and a negligibly small additive cost to the number of logical qubits. We expect that twist-based lattice surgery has it own associated costs, which may exceed those of our twist-free approach, but twist performance has never been fully quantified and so represents a currently unknown factor in quantum computing design.

In Sec. V, we show how to reduce algorithm run times using a technique that we call temporal encoding of lattice surgery. By using fast lattice-surgery operations (which are inevitably noisier), errors arising from the extra noise can be corrected by encoding the sequence of measured Pauli operators within a classical error-correcting code. The resulting run-time improvement grows (as a multiplicative factor) with the parallelizability of the algorithm and the total algorithm run time. We find that in a regime of interest to quantum algorithms of a practical scale, we can achieve a 2.2× run-time improvement. Our temporal encoding does not directly lead to additional qubit overhead costs since it occurs in the time domain and so the overall space-time complexity is improved.

Lastly, in Sec. VI, we describe our core-cache architecture. We show that by using thin rectangular strips of surface codes for settings where a large noise bias is present, the overhead costs due to routing in our proposed architecture adds a multiplicative factor of 1.5 increase to the total resource costs for performing lattice surgery. This can be compared with the factor-of-2 cost of Litinski's fast data-access structures [12]. Furthermore, we provide a layout that compactly stores surface-code patches in a cache to further reduce the extra overhead arising from routing costs, at the cost of some additional time needed for reading from and writing to the cache. Using the numerical results obtained in Sec. III, in Appendix C we provide resource-cost estimates for simulating the Hubbard model using our core-cache architecture.

II. BRIEF REVIEW OF UNIVERSAL QUANTUM COMPUTING VIA LATTICE SURGERY

In Sec. II A, we briefly review the principles of Pauli-based computation (PBC) used throughout this work. We then review in Sec. II B how multiqubit Pauli operators are measured using lattice surgery.

A. Overview of PBC

In the model of PBC, we have a reserve of magic states and we drive the computation by performing a sequence of multiqubit Pauli measurements $\{P_1, P_2, \ldots, P_\mu\}$, where later Pauli measurements depend on measurement outcomes of earlier measurements. In this notation, P_2 does not denote a specific Pauli but one conditional on the outcome of P_1. This conditionality occurs because (in the circuit picture) each Pauli measurement would be followed by a conditional Clifford operation. However, in a PBC, these Cliffords are conjugated to the end of the computation, thereby changing subsequent Pauli measurements. Since in a PBC all Cliffords are performed "in software," it is clear that the algorithm run time will be independent of the Clifford complexity. The idea of PBC appears throughout the literature but the phrase "Pauli-based computation" was coined in Ref. [29]. In Fig. 2, we present several computationally equivalent circuit diagrams for performing 2 T gates, with the last diagram representing the PBC approach.

In Sec. II B, we review how multiqubit Pauli measurements can be performed using lattice surgery. Crucially, even when Pauli operators commute, it might not be

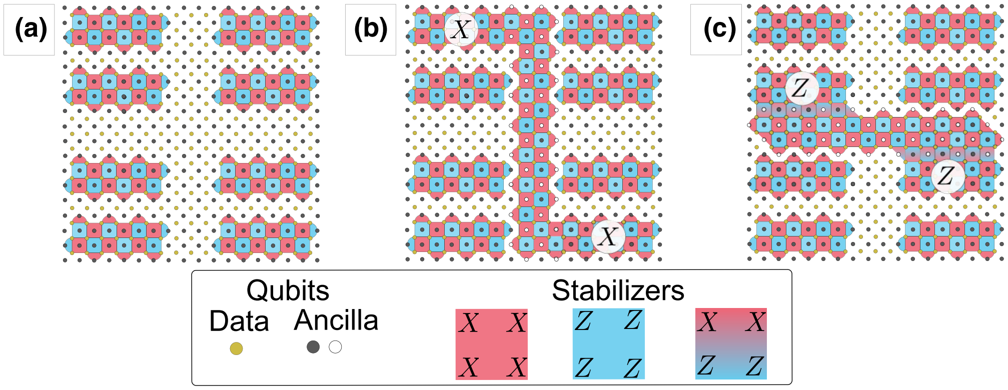

FIG. 2. A simple example of lattice surgery, illustrating the space overhead needed for routing. (a) Eight rectangular surface-code patches with some surrounding idle qubits that form the routing space. (b) The stabilizers needed to measure a logical $X \otimes X$ operator between the separated surface-code patches using the routing space between them. (c) Similarly, the stabilizers needed to measure a logical $Z \otimes Z$ operator between the separated surface-code patches using the available routing space. Stabilizers of mixed X and Z type are referred to as domain walls and are used to ensure that the d_z distance of the surface code does not decrease when measuring Z-type logical Pauli operators via lattice surgery. White ancillas mark the stabilizers that directly contribute toward computing the parity of the $X \otimes X$ and $Z \otimes Z$ measurement outcomes.

possible to measure such Pauli operators simultaneously due to the extra space required to perform lattice surgery (known as the routing space). We can be obstructed from measuring commuting Pauli operators when the required lattice-surgery operations need access to the same routing space. Therefore, it is appropriate to consider sequentially measuring each Pauli operator, which we call a sequential Pauli-based computation (seqPBC). In seqPBC, the time required to execute all the Pauli measurements is then proportional to $T_{\mathrm{PBC}} = (d_m + 1)\mu$, where we budget $+1$ for resetting qubits between lattice-surgery operations. Here, d_m corresponds to the number of rounds of stabilizer measurements during lattice surgery and μ is the number of sequential Pauli operators being measured. The proportionality factor depends on the time required to measure the surface-code stabilizers during one syndrome-measurement round.

There are several contributions to the number of Pauli measurements μ. At a high level of the stack, we may think of a quantum algorithm as consisting of a series of unitaries with some Pauli measurements for readout and we may let N_A denote the number of such algorithmic readout measurements. However, as we see in Fig. 2, non-Clifford unitaries are performed by measurements. If an algorithm has N_T T gates, then we also need an additional N_T Pauli measurements. The Clifford plus T gate set is universal. However, it is advantageous to use an overcomplete gate set such as Clifford plus T and Toffoli. While Toffoli can be synthesized using four T gates [30], it is often more efficient to directly prepare Toffoli magic states [30–33]. Furthermore, it only takes three Pauli measurements to teleport a Toffoli state rather than the four measurements needed to teleport T states and then synthesize a Toffoli. As such, if an algorithm can be executed with N_{TOF} Toffoli gates and N_T T gates, then we need $N_T + 3N_{\mathrm{TOF}}$ measurements to perform these teleportations. Further refinements are possible by using an even richer gate set and preparing more exotic states [34–36] but we do not explicitly discuss those schemes here. Lastly, we can replace some non-Clifford gates with Pauli measurements and feedforward (with no magic state needed). For instance, such a measurement appears in Gidney's circuits for adders [37] and, more generally, any uncomputation subroutine of an algorithm, where the Toffoli gates are replaced with Pauli measurements and feedforward. We use N_{unTOF} to denote the number of Toffoli uncomputations performed in this manner. Hence, for a Clifford plus T and Toffoli gate set, the total number of Pauli measurements is $\mu = N_A + N_T + 3N_{\mathrm{TOF}} + N_{\mathrm{unTOF}}$. Note that in many algorithms, Toffolis exclusively appear in compute-uncompute pairs and then we have $N_{\mathrm{unTOF}} = N_{\mathrm{TOF}}$. Furthermore, algorithms often only have a small number of qubit readouts, so $N_A \ll N_T, N_{\mathrm{TOF}}$. As such, we commonly have $\mu \approx N_T + 4N_{\mathrm{TOF}}$.

An architecture also requires time T_{magic} to produce the required magic states—say, N_T T-states and N_{TOF} Toffoli states. If an architecture produces all the required magic states in a shorter amount of time than is required to teleport them, so that $T_{\mathrm{magic}} < T_{\mathrm{PBC}}$, then we say that the seqPBC is Clifford bottlenecked. On the other hand, if $T_{\mathrm{magic}} \geq T_{\mathrm{PBC}}$, we say that it is magic-state bottlenecked. The running time of the algorithm is determined by $\max\{T_{\mathrm{magic}}, T_{\mathrm{PBC}}\}$.

It is informative to briefly review the impact of these bottlenecks on the history of algorithm resource analysis. The time T_{magic} can be made arbitrarily small,

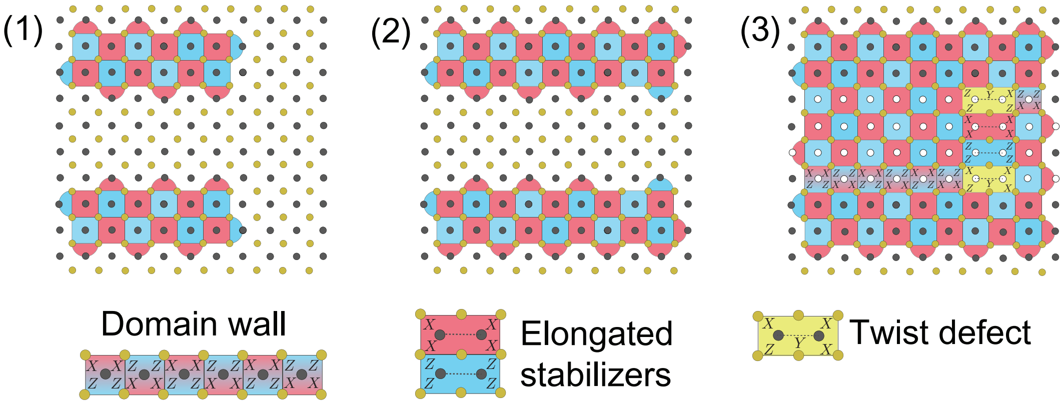

FIG. 3. A simple example of a lattice-surgery protocol using a twist defect in the bulk of the lattice to measure a $Y \otimes Y$ logical operator. Step (1) is the initial setup of two surface-code patches. In step (2), the surface-code patches are extended and corners moved using the routing space. Such an extension allows logical Y operators of the surface code to be expressed along a horizontal boundary. In step (3), we measure $Y \otimes Y$ using a combination of domain walls, elongated stabilizers, and two twist defects. The domain walls measure $Z \otimes Z \otimes X \otimes X$ stabilizers and typically offer no additional challenge compared to normal stabilizer measurements. The elongated stabilizers are long-range operators that pose an additional difficulty in hardware implementations. Elongated stabilizers can be implemented using either two ancilla qubits [e.g., prepared in a Greenberger-Horne-Zeilinger (GHZ) state], that we connect with a dashed line, or these two ancilla qubits necessarily merged into a single qubit (hardwired in the architecture) and using long-range gates. Twist defects (yellow plaquettes) present the biggest difficulty, since in addition to the challenges faced by elongated stabilizers, they also require a weight-5 measurement.

simply by increasing the number of magic state factories, although this comes at an increased qubit cost. Some early algorithm-overhead estimates [5,38,39] minimized T_{magic} by having a large number of factories, leading to widespread claims that magic state factories could be approximately 99% of the whole device. However, such resource estimates ignore the question of how quickly these states can be teleported and ignored the T_{PBC} bottleneck. Accounting for this bottleneck, there is no benefit from pushing T_{magic} to be small $T_{\text{magic}} \ll T_{\text{PBC}}$. As such, more recent and careful overhead analyses [33,40–42] have assumed only a few factories, which is enough to achieve $T_{\text{magic}} \approx T_{\text{PBC}}$ and results in only a small percentage of the device footprint (often less than 1%) being used as a magic state factory. While these analyses have minimal qubit cost, the run time is now T_{PBC} bottlenecked, motivating rigorous approaches to beating the T_{PBC} bottleneck. Later, in Sec. V A, we review prior lattice-surgery methods to speed up algorithms by using additional teleportation gadgets, although this comes at a high qubit cost. We then introduce our own approach that instead uses temporal encoding of lattice surgery (TELS).

B. Overview of lattice surgery

For quantum hardware where physical qubits can only interact with one another locally, lattice surgery is a fault-tolerant protocol that can be used to measure arbitrary multiqubit Pauli operators. The main idea is to encode the logical qubits in some topological code arranged in a 2D layout (in this work, all logical qubits are encoded in the rotated surface code [43]). The layout contains extra routing space between the surface-code patches, which consist of additional qubits. By applying the appropriate gauge-fixing operations in the routing space (see, e.g., Ref. [44]), which involves measuring surface-code stabilizers, the surface-code patches involved in the Pauli measurement are merged into one larger surface-code patch. After gauge fixing, the parity of the measurement outcome of the multiqubit Pauli operator being measured is obtained by taking the products of the appropriate stabilizers in the routing space. Lastly, the surface-code patches for each logical qubit can be detached from the merged patch by measuring the qubits in the routing space in the appropriate basis. An illustration of $X \otimes X$ and $Z \otimes Z$ Pauli measurements is shown in Fig. 3. Products of the surface-code stabilizers marked by white ancilla qubits give the parity of the $X \otimes X$ and $Z \otimes Z$ measurement outcomes. Note that for Z-type Pauli measurements, we use domain walls at the Z logical boundaries of the surface-code patches. Domain walls correspond to stabilizers of mixed X and Z type, as illustrated in Fig. 1. The primary reason for using domain walls is to prevent a reduction in minimum-weight representatives of logical Z operators during lattice surgery.

To measure multiqubit Pauli operators containing Y terms, one option is to extend the surface-code patches using the routing space in such a way that the logical Y operators can be expressed along horizontal boundaries of the surface code. Logical Y operators can then be measured using a twist defect, as shown in Fig. 1. Note that such a protocol requires measuring a weight-5 operator, such as the ones shown in yellow in Fig. 1. Such high-weight measurements can be undesirable for many hardware architectures. An alternative approach that does not require the extension of surface-code patches and the use of twist defects in the bulk is provided in Sec. IV.

When performing a lattice-surgery measurement of a logical Pauli operator, there will be some probability that we obtain the wrong outcome. Even with large code distances, the lattice-surgery measurement could still fail due to timelike errors occurring during the finite time allowed for lattice surgery. The probability of these failure events is exponentially suppressed in the number of rounds d_m for which we repeat the stabilizer measurements during lattice surgery. Therefore, we call d_m the measurement distance, which quantifies the protection against repeated measurement failures during lattice surgery, and hence explains our choice of subscript. Defining d_z and d_x to be minimum weights of logical Z- and X-type operators of the surface code, this exponential suppression will hold until $d_m \gg O(d_z, d_x)$, when logical Pauli errors become the dominate mechanism again. Let us assume that code distances d_x and d_z are chosen so that even a single logical Z and X error is very unlikely over the course of the whole computation. We expect, and numerically find, that these timelike errors occur with a probability $\mathbb{P}$ for which we have a bound of the form

$$\mathbb{P} \leq La(pb)^{c(d_m+1)}, \tag{1}$$

where p quantifies the physical gate-failure probabilities, $\{a, b, c\}$ are constants, and L is the area of the patch used for lattice surgery. The value of L will vary for different measurements and different layouts but it will be convenient to think of it as a constant representing the worst-case (or average) area of lattice-surgery patches.

In general, if we want to sequentially perform μ Pauli measurements in the algorithm and we want them to fail with probability no more than δ, then we choose d_m to be large enough such that

$$\mu La(pb)^{c(d_m+1)} \approx 1 - (1 - \mathbb{P})^{\mu} \leq \delta, \tag{2}$$

where the approximation holds for small $\mathbb{P}$. In Sec. III, after introducing a decoder compatible with lattice-surgery protocols, we compute timelike failure probabilities in addition to probabilities for other noise processes given a biased circuit-level noise model. Such results allow us to obtain accurate resource overhead estimates for implementing quantum algorithms and are discussed further in Sec. VI.

III. DECODING TIMELIKE ERRORS DURING LATTICE SURGERY

In this section, we provide an explicit decoding protocol for correcting both spacelike and timelike errors that can occur during lattice-surgery protocols. In particular, our protocol protects logical qubits encoded in surface-code patches while at the same time correcting logical multiqubit Pauli-measurement failures that can occur during lattice surgery. We then provide numerical results for performing $X \otimes X$ measurements, showing both the logical multiqubit Pauli-measurement failure rate as a function of the number of syndrome-measurement rounds and the logical qubit failure rates.

The generalization of toric code decoders with periodic boundary conditions to the surface code has been done in Refs. [7,45,46] by adding virtual vertices at the boundaries of the surface code. Follow-up work in Refs. [8,9] has provided a high-level account of how surface-code decoders can be used in the context of lattice surgery. However, details such as the correct specification of boundary vertex locations for both spacelike and timelike failures have not been provided. In Sec. III A, our lattice-surgery decoding algorithm makes use of boundary vertices for both spatial and timelike boundaries. Further, no previous work has performed such realistic circuit-level simulations of lattice surgery. For comparison, in Ref. [33], timelike errors have been simulated, but the authors have used a toy model with idealized boundaries to exclude logical spacelike errors and thereby simplify both the simulations and the required decoding algorithm. In what follows, we refer to a surface-code patch encoding a logical qubit as *a logical patch*. The space used for performing multiqubit Pauli measurements via lattice surgery is referred to as the *routing space* or *routing region*.

In Fig. 4, we provide an example of an $X \otimes X$ Pauli measurement performed between two $d_x = 5$, $d_z = 7$ logical patches. After preparing ancilla qubits (gray vertices) in the routing space in $|+\rangle$ and data qubits (yellow vertices) in $|0\rangle$, X-type surface-code stabilizers are measured. The products of stabilizers marked by white vertices gives the measurement outcome of the logical $X \otimes X$ operator. In what follows, white vertices the product of which gives the result of a $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ Pauli measurement are referred to as *parity vertices*. We say that a logical timelike failure occurs if a set of errors result in the wrong parity measurement of $P_1 \otimes P_2 \otimes \cdots \otimes P_k$. Further, we assume that the logical patches are measured for r rounds prior to being merged in round $r + 1$.

When measuring $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ using lattice surgery, in addition to an odd number of measurement errors occurring in round $r + 1$, an odd number of data-qubit errors along the boundaries of the logical patches prior to the merge can also result in a logical timelike failure. An example is provided in Fig. 4, where a single data-qubit Z error along a boundary of the left logical patch prior to the merge gives the wrong parity of $X \otimes X$. We also note that during the syndrome-measurement round $r + 1$, an odd number of data-qubit errors that anticommute

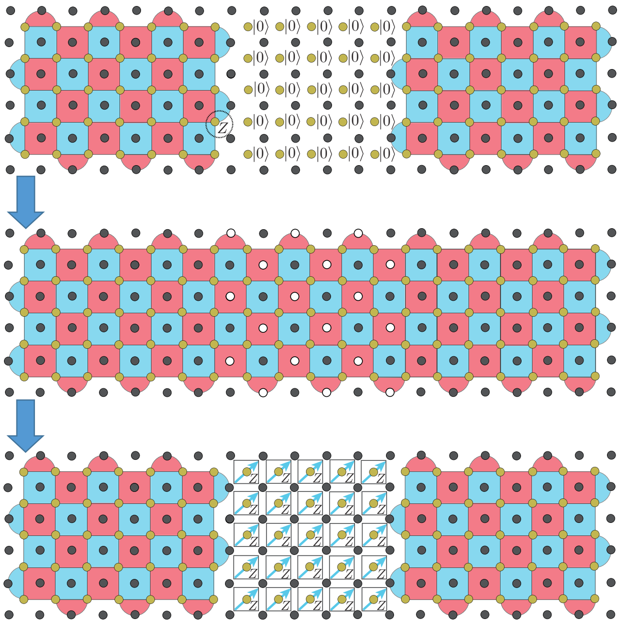

FIG. 4. An example of an $X \otimes X$ measurement performed on two $d_x = 5$, $d_z = 7$ logical patches. Data qubits in the routing space are first prepared in the $|0\rangle$ state. In the first round of stabilizer measurements, where the logical patches are merged into one large surface-code patch, the product of all stabilizers marked by white vertices (which we call parity vertices) gives the result of the $X \otimes X$ measurement. After measuring the stabilizers of the merged patch for d_m rounds, the patches are split by measuring the data qubits in the routing region in the Z basis. In the first round of the merge, measurement errors occurring on parity vertices can result in a wrong $X \otimes X$ measurement outcome. Additionally, an odd number of data-qubit Z errors along the boundary of the logical patches prior to the merge, such as the one circled in the top row of the figure, can also result in a wrong $X \otimes X$ measurement outcome.

with $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ will (unless corrected) also result in a logical timelike failure.

The above examples show that in order to obtain the correct parity measurement of a $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ operator in the presence of full circuit-level noise, one must have a decoding scheme that, while constantly correcting errors on the logical patches, also corrects spacelike and timelike errors that can flip the parity of the measurement outcome.

A. The decoding algorithm

In order to correct logical timelike failures using a minimum-weight-perfect-matching (MWPM) decoder [47], we must add timelike boundaries to the matching graphs of the surface code as shown in Fig. 5. In particular, we divide the measurement of an operator $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ into three steps. In the first step, the logical patches are measured for r rounds. In round $r + 1$, the patches are merged by measuring the appropriate operators in the routing space (see, e.g., Fig. 4) and the parity of the measurement outcome is given by the product of all parity vertices. The merged patches are measured for d_m rounds and then, in round $d_m + 1$, the qubits in the routing space are measured in the appropriate basis to split the patches back to their original configuration. In round r, we add extra virtual vertices to the matching graph with vertical edges that are incident to such vertices and to the parity vertices in round $r + 1$ (see the pink edges in Fig. 5(b) for the $X \otimes X$ measurement). We call such vertices *past ancilla vertices* and the pink edges incident to them *past vertical edges*. Similarly, in round $r + d_m + 1$ (i.e., right after the split), we add virtual vertices to the matching graph with vertical edges that are incident to such vertices

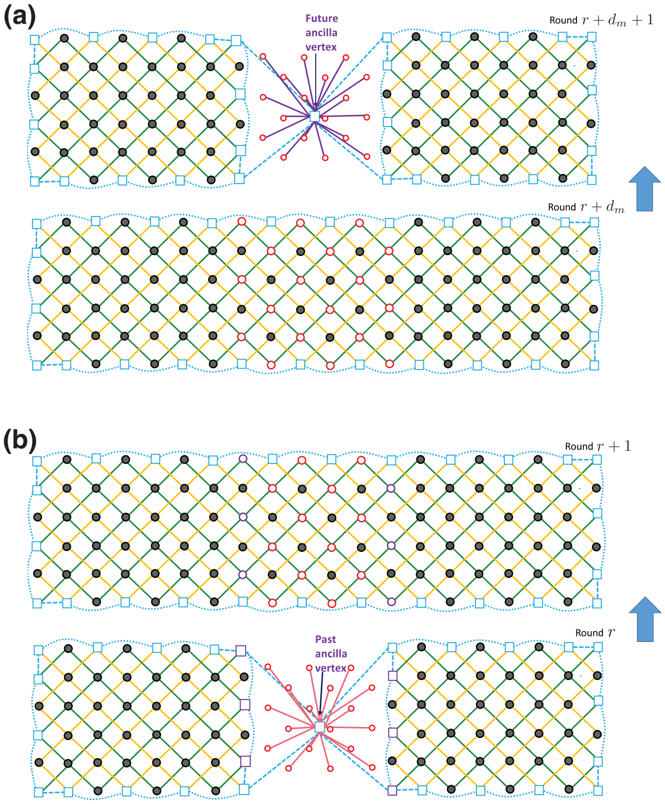

FIG. 5. Various 2D slices of the matching graph used for performing an $X \otimes X$ Pauli measurement via lattice surgery. (a) The 2D slices of the surface-code matching graphs for syndrome-measurement rounds $r + d_m$ (bottom) and $r + d_m + 1$ (top). The graph in round $r + d_m + 1$ includes the future ancilla vertex, with future vertical edges (solid purple edges) connecting to the parity vertices (white vertices circled in red) of the matching graph in round $r + d_m$. (b) The 2D slices of the surface-code matching graphs for syndrome-measurement rounds r (bottom) and $r + 1$ (top). The graph in round r includes the past ancilla vertex with past vertical edges (solid pink edges) connecting to the parity vertices (white vertices circled in red) of the matching graph in round $r + 1$. The transition vertices are the purple boundary vertices of the graph prior to the merge (note that transition vertices appear in rounds 1 to r), in addition to the purple parity vertices in round $r + 1$.

and to the parity vertices in round $r + d_m$ (see the purple edges in Fig. 5(a) for the $X \otimes X$ measurement). We call such vertices *future ancilla vertices* and the purple edges incident to them *future vertical edges*. Importantly, the pink vertical edges that are incident to the past ancilla vertices and to the parity vertices in round $r + 1$ have zero weight, while the purple vertical edges incident to the parity vertices in round $r + d_m$ and the future ancilla vertices have nonzero weights. These weights are computed from all timelike failure processes, which can result in measurement errors occurring in round $r + d_m$. When performing MWPM over the full syndrome history, the parity of the measurement outcome of $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ is flipped if there are an odd number of highlighted vertical edges incident to parity vertices in rounds $r + 1$ and $r + 2$. In such a setting, one would require a sequence of consecutive measurement errors that is greater than $(d_m - 1)/2$ in order to cause a timelike logical failure. Note that since the measurement outcomes of the parity vertices in round $r + 1$ are random, such vertices are never highlighted. If a change in the measurement outcomes of a subset of the parity vertices are observed between rounds $r + 1$ and $r + 2$, then vertices corresponding to such parity vertices in round $r + 2$ would be highlighted. Furthermore, green horizontal edges incident to the parity vertices in round $r + 1$ are taken to have zero weight (or can be omitted).

As mentioned above, a lattice-surgery decoder also needs to correct logical timelike failures arising from sets of data qubit errors along boundaries of the logical qubit patches prior to merging them (recall the example shown in Fig. 4). The decoder also needs to correct wrong parity measurements arising from data qubit errors in round $r + 1$ that anticommute with the Pauli operator being measured by lattice surgery. In constructing such a decoder, note that prior to merging the logical patches for the $X \otimes X$ measurement, a single Z error along the relevant boundaries would result in a highlighted edge (after implementing MWPM over the full syndrome history) incident to one of the purple boundary vertices shown in Fig. 5(b). Note that such boundary vertices become parity vertices after merging the surface-code patches. For the measurement of a general $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ Pauli operator, we define *transition vertices* to be vertices in the set $V_{\text{Bd}}^{(s)} = \{v_{b_1}^{(s)}, \ldots, v_{b_m}^{(s)}\}$, where $1 \leq s \leq r + 1$. When $s < r + 1$, $\{v_{b_1}^{(s)}, \ldots, v_{b_m}^{(s)}\}$ are labels for boundary vertices of the graphs of split logical patches that become parity vertices in round $r + 1$. If $s = r + 1$, then $V_{\text{Bd}}^{(r+1)}$ is the set of parity vertices along the boundaries of the logical qubit patches and routing space used to merge the logical qubits (see, e.g., the parity vertices highlighted in purple in Fig. 6). Based on previous observations, after implementing MWPM over the full syndrome history of a multiqubit Pauli measurement via lattice surgery, if an odd number of spacelike highlighted edges are present in logical patches and such edges are incident to transition vertices, the parity of the Pauli measurement needs to be flipped. An illustration of a 2D slice of the matching graphs of Fig. 5 in the timelike direction (which contains a subset of the spacelike edges and vertices) is shown in Fig. 6. In particular, the figure illustrates transition vertices and the past and future ancilla vertices in addition to the past and future vertical edges.

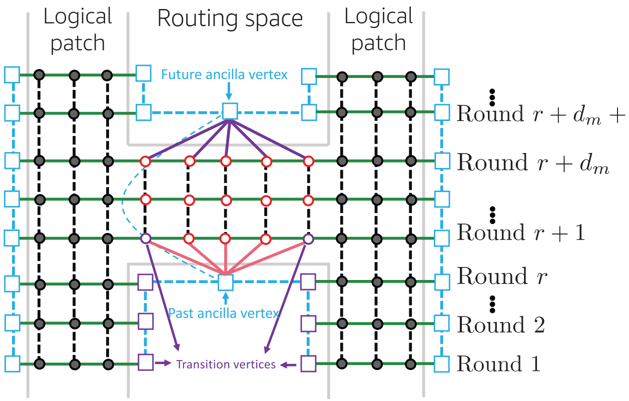

FIG. 6. A 2D slice of the surface-code matching graph in the timelike direction for syndrome-measurement rounds performed during an $X \otimes X$ measurement protocol using lattice surgery. The vertical axis corresponds to the timelike direction. We show a subset of the vertices and edges of the matching graph for correcting Z errors for an $X \otimes X$ measurement using lattice surgery. The parity vertices are the white and red vertices in the ancilla patch region. The transition vertices are both boundary and parity vertices colored in purple. The pink edges are past vertical edges incident to the past ancilla vertex and the parity vertices in round $r + 1$, whereas the purple edges are future ancilla edges incident to the parity vertices in round $r + d_m$ and the future ancilla vertex. We also add a dashed-blue weightless edge connecting the past and future ancilla vertices.

Combining all the notions introduced in this section, the decoding algorithm for implementing a multiqubit Pauli measurement via lattice surgery is described in Algorithm 1. Each highlighted edge in step (7) of Algorithm 1 encodes a particular data qubit correction. Writing such corrections as a binary row vector, where each column corresponds to a data qubit, we add all corrections arising from each highlighted edge using modulo-2 arithmetic. Further, note that space-time correlated edges incident to parity vertices in round $r + 1$ need to be treated with care in order to correct errors up to the full code distance. In particular, a subset of the space-time correlated edges incident to transition vertices can also contribute to v_2 (defined in Algorithm 1). A more careful treatment of such edges is provided in Appendix B.

B. Decoder simplifications

We point out a simplification that can be made in the implementation of Algorithm 1. Note that vertices in $V_{\text{par}}^{(r+1)}$

```
Result: Data qubit and parity measurement corrections.
initialize: $v_1 = v_2 = 0$. Let $G_r$ be the graph for the
  surface code patches before, during and after the
  merge.
Measurement: Measure the stabilizers of the split
  logical patches for $r$ rounds. Merge the logical patches
  in round $r + 1$ via lattice surgery to perform the
  $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ measurement and let $s_{\text{par}}$ be the
  parity of the measurement outcome. Repeat the
  stabilizer measurements of the merged patch for
  $d_m - 1$ rounds.
1) Add the past ancilla vertex $v_{\text{past}}$ to $G_r$ for the round $r$
  (round before the merge). Let
  $V_{\text{par}}^{(r+1)} = \{v_{\text{par}}^{(1)}, \cdots, v_{\text{par}}^{(k)}\}$ be the set of parity
  vertices for the syndrome measurement round $r + 1$.
  Add weightless past vertical edges to $G_r$ which are
  incident to $v_{\text{past}}$ and all vertices $v \in V_{\text{par}}^{(r+1)}$. Add
  weightless edges to $G_r$ which are between $v_{\text{past}}$ and
  virtual boundary edges of all surface code patches
2) Add the future ancilla vertex $v_{\text{future}}$ to $G_r$ for the
  round $r + d_m + 1$ (round after the merge). Let
  $V_{\text{par}}^{(r+d_m)} = \{\tilde{v}_{\text{par}}^{(1)}, \cdots, \tilde{v}_{\text{par}}^{(k)}\}$ be the set of parity
  vertices for the syndrome measurement round $r + d_m$.
  Add future vertical edges (of non-zero weight) to $G_r$
  which are incident to $v_{\text{future}}$ and all vertices
  $v \in V_{\text{par}}^{(r+d_m)}$
3) Add a weightless edge to $G_r$ which is incident to
  $v_{\text{past}}$ and $v_{\text{future}}$
4) Set all edges incident to any two vertices
  $v_i, v_j \in V_{\text{par}}^{(r+1)}$ to have zero weight
5) Given the full syndrome measurement history, if the
  total number of highlighted vertices (obtained by
  taking the difference between any two consecutive
  syndrome measurement rounds modulo 2) is odd,
  highlight $v_{\text{future}}$
6) Implement MWPM on $G_r$. Set $v_1$ to be the number
  of highlighted edges incident to vertices in $V_{\text{par}}^{(r+1)}$ and
  $V_{\text{par}}^{(r+2)}$, and $v_2$ to be the number of highlighted edges
  incident to transition vertices in the data-qubit patch
  regions. If $v_1 + v_2$ is odd, set $s_{\text{par}} \to s_{\text{par}} + 1$
  $\pmod 2$
7) Apply data qubit corrections based on all highlighted
  two-dimensional and space-time correlated edges.
```

Algorithm 1. The decoding algorithm for measuring $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ via lattice surgery.

are never highlighted during MWPM due to the random outcomes of stabilizers in the routing space marked by white vertices in round $r + 1$. As such, one could remove all vertices in $V_{\text{par}}^{(r+1)}$ and instead have the past vertical edges incident to the vertices in $V_{\text{par}}^{(r+2)}$. In such a setting, edges incident to $V_{\text{par}}^{(r+1)}$ and $V_{\text{par}}^{(r+2)}$ would be removed and their weights would be assigned to the past vertical edges. Lastly, we remark that all boundary vertices, including the past and future ancilla vertices, can be merged into a single boundary vertex. In such a setting, each edge of the matching graph G_r encodes both a spacelike *and* timelike correction. The timelike component for the edge e_j is obtained by observing whether the failure mechanism resulting in the highlighted edge e_j flips the parity of the multiqubit Pauli measurement. We choose to describe the decoding protocol using Algorithm 1 to avoid figures with multiple edges all incident to the same boundary vertex.

C. Noise model and simulation methodology

Using the decoding algorithm given in Algorithm 1, we perform a full circuit-level noise simulation of various code distances and syndrome-measurement rounds to estimate the parameters in Eq. (1) for a $X \otimes X$ measurement. We choose the following biased circuit-level noise model:

(1) Each single-qubit gate location is followed by a Pauli Z error with probability $p/3$ and Pauli X and Y errors each with probability $p/3\eta$.

(2) Each two-qubit gate is followed by a $\{Z \otimes I, I \otimes Z, Z \otimes Z\}$ error with probability $p/15$ each and a $\{X \otimes I, I \otimes X, X \otimes X, Z \otimes X, Y \otimes I, Y \otimes X, I \otimes Y, Y \otimes Z, X \otimes Z, Z \otimes Y, X \otimes Y, Y \otimes Y\}$, each with probability $p/15\eta$.

(3) With probability $2p/3\eta$, the preparation of the $|0\rangle$ state is replaced by $|1\rangle = X|0\rangle$. Similarly, with probability $2p/3$, the preparation of the $|+\rangle$ state is replaced by $|-\rangle = Z|+\rangle$.

(4) With probability $2p/3\eta$, a single-qubit Z basis measurement outcome is flipped. With probability $2p/3$, a single-qubit X-basis measurement outcome is flipped.

(5) Lastly, each idle gate location is followed by a Pauli Z with probability $p/3$ and a $\{X, Y\}$ error, each with probability $p/3\eta$.

In our simulations, we choose $\eta = 100$ and for simplicity add a single idle location on the data qubits during the measurement and reinitialization of ancilla qubits. Note that in the limit $\eta \to 1$, the above noise model reduces to the depolarizing noise model used in Refs. [48,49], with the exception that two idle locations are included during measurement and reinitialization of the ancillas. Furthermore, the above noise model assumes that the duration of a controlled-NOT (CNOT) gate is identical to the duration of an ancilla measurement and reinitialization.

For a biased circuit-level noise model (such as the one described above) and an $X \otimes X$ Pauli measurement implemented via lattice surgery, there are 15 different types of failure mechanisms that can arise during the protocol. We label such failure mechanisms using the binary string $(b_{\text{ZL}}, b_{\text{TL}}, b_{\text{ZR}}, b_X)$. The bit $b_{\text{ZL}} = 1$ corresponds to a logical Z error on the left logical patch, whereas $b_{\text{ZR}} = 1$

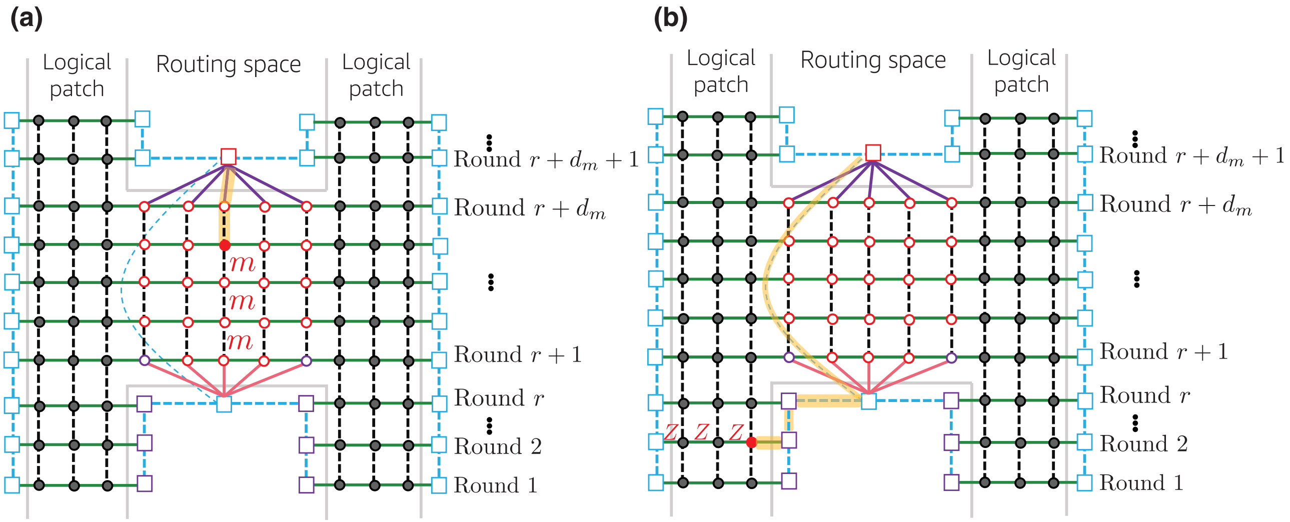

FIG. 7. (a) An example of a series of consecutive measurement errors on the same parity vertex located in the routing space resulting in a $(0, 1, 0, 0)$ logical failure for an $X \otimes X$ measurement. (b) An example of a horizontal string of spacelike Z data-qubit errors on the left logical patch resulting in a $(1, 1, 0, 0)$ logical failure.

corresponds to a logical Z error on the right logical patch. The bit $b_{\mathrm{TL}} = 1$ indicates a logical timelike failure. Finally, the bit b_X indicates whether a logical X error occurs during the lattice-surgery protocol. Examples of such failure mechanisms using 2D slices of the matching graph are shown in Fig. 7. For instance, in Fig. 7(a), a series of consecutive measurement errors occur on the same parity vertex, starting in round $r + 1$. Since a single measurement error occurs in round $r + 1$, the wrong parity of $X \otimes X$ is measured. Due to the series of measurement errors, a single parity vertex near the top boundary is highlighted in G_r. The shortest path correction (highlighted in yellow) matches the highlighted parity vertex to the future ancilla vertex. Hence no parity corrections are applied and a logical $(0, 1, 0, 0)$ error occurs.

Another example is shown in Fig. 7(b), where a string of Z data-qubit errors results in the highlighted vertices shown in the figure. Prior to the merge, there are no Z errors at the boundary between the logical patches and routing space. Therefore, the correct parity of $X \otimes X$ is measured. However, the minimum-weight path (highlighted in yellow) connecting the highlighted vertex to the future ancilla vertex goes through a transition vertex, so that $v_2 = 1$. The correction thus results in a logical Z error on the left logical patch, in addition to a logical parity measurement failure (since the decoder incorrectly flips the parity) leaving the code with a logical $(1, 1, 0, 0)$ error. Additional examples of failure mechanisms using spacetime diagrams instead of matching graphs are provided in Fig. 8.

We note that if the stabilizer measurements are terminated after $r + d_m$ rounds, higher-order failure mechanisms are required to produce logical failures corresponding to $(1, 0, 0, 0)$, $(0, 0, 1, 0)$, $(1, 0, 1, 0)$, and $(1, 1, 1, 0)$ bit strings. Failures corresponding to such strings are thus much less likely compared to $(1, 1, 0, 0)$, $(0, 1, 0, 0)$, and $(0, 1, 1, 0)$ [50]. For instance, to obtain a logical failure of the type $(1, 0, 0, 0)$, a logical error on the left logical patch would need to occur without flipping the parity of the $X \otimes X$ measurement. As such, in addition to a logical Z error occurring before the merge, a second failure mechanism would need to occur to undo the wrong parity flip [such as a string of measurement errors like the one shown in Fig. 7(a)]. Given these observations, we only present the logical failure-rate polynomials $\mathbb{P}_{(0,1,0,0)}$, $\mathbb{P}_{(1,1,0,0)}$, and $\mathbb{P}_{(0,1,1,0)}$. Note that due to the high noise bias, we choose a d_x distance such that $\mu\mathbb{P}_{(0,0,0,1)} \leq \delta$, where μ is the number of lattice-surgery operations in the algorithm.

Our simulations are performed for syndrome-measurement rounds 1 to $r + d_m$, based on the biased circuit-level noise model described above. The last round is a round of perfect error correction to guarantee projection to the code space. We use Algorithm 1 to correct both spacelike and timelike errors, where each edge in step (7) of the algorithm encodes a particular correction on a subset of the data qubits. Note that we do not perform a round of perfect error correction between rounds r and $r + 1$, as was done in Ref. [44]. Instead, we perform MWPM using the full syndrome-measurement history from rounds 1 to $r + d_m$, and use Algorithm 1 to determine which corrections are applied.

D. Simulation results and conclusions

Here, we report the outcome of our lattice-surgery simulations as summarized by Eqs. (3)–(5) and Fig. 9. For each

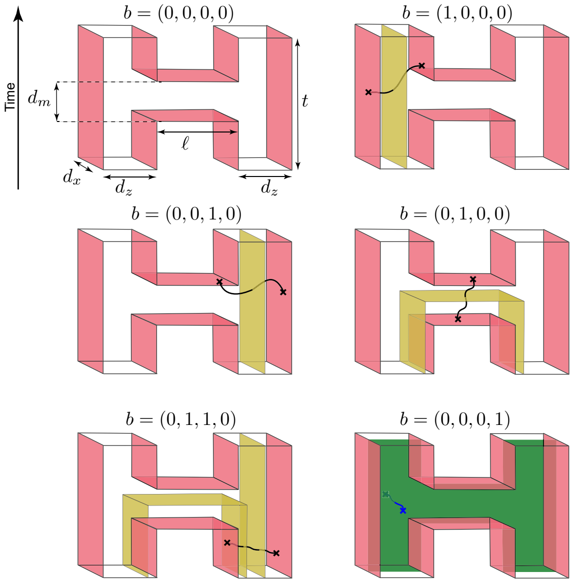

FIG. 8. A space-time diagram of a lattice-surgery protocol, illustrating different types of failure mechanisms represented by the vector $b = (b_{\mathrm{ZL}}, b_{\mathrm{TL}}, b_{\mathrm{ZR}}, b_X)$. For each element b_j in the vector b, we associate a logical sheet (shown in yellow or green when triggered) and we have $b_j = 1$ if and only if the relevant error type crosses the logical sheet an odd number of times. The Z strings trigger yellow logical sheets and terminate on pink boundaries. The X strings trigger green logical sheets and terminate on the foreground and background boundaries (these are transparent for visual clarity). For instance, a $(1,0,0,0)$ event (top right) occurs in the presence of a logical Z stringlike excitation on the left surface-code patch. Since the excitation does not cross the yellow ⊓-shaped logical sheet, the correct outcome of the multiqubit Pauli measurement is recorded. A $(0,1,0,0)$ event (middle right) is a pure timelike failure, where the incorrect multiqubit Pauli-measurement outcome is recorded without introducing additional logical Z failures to the two logical patches. This holds because the Z string only crosses the yellow ⊓-shaped logical sheet. A $(0,1,1,0)$ event (bottom left) occurs when a Z stringlike excitation crosses the rightmost yellow logical sheet in addition to the yellow ⊓-shaped logical sheet. Such an error results in both a logical Z error on the right logical patch in addition to a timelike lattice-surgery failure. Lastly, we illustrate an X stringlike excitation crossing the green logical sheet, resulting in a logical X failure on both logical patches.

of the dominant failure mechanisms in $(b_{\mathrm{ZL}}, b_{\mathrm{TL}}, b_{\mathrm{ZR}}, b_X)$, we fit all our data to an ansatz with two free parameters to generate the failure-rate polynomials given by

$$\mathbb{P}_{(0,1,0,0)} = 0.01634 d_x \ell (21.93p)^{(d_m+1)/2}, \tag{3}$$

$$\mathbb{P}_{(1,1,0,0)} = 0.03148 d_x (28.91p)^{(d_z+1)/2}, \tag{4}$$

$$\mathbb{P}_{(0,1,1,0)} = 0.03 d_x (28.95p)^{(d_z+1)/2}, \tag{5}$$

$$\mathbb{P}_{(0,0,0,1)} = 0.0148 d_z (0.762p)^{(d_x+1)/2}. \tag{6}$$

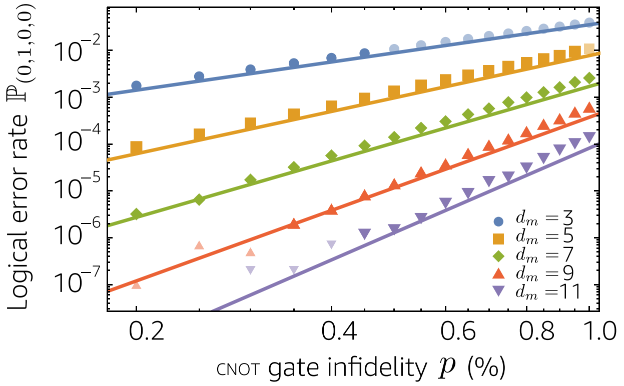

FIG. 9. A comparison between the best-fit polynomial $\mathbb{P}_{(0,1,0,0)}$ for various values of d_m given in Eq. (3) with the data obtained from our Monte Carlo simulations. We choose parameters where $d_x = 9$, $d_z = 11$, $r = d_z$, and $\ell = 5$. The translucent data points are omitted when obtaining the fitting polynomial in Eq. (3).

In Eq. (3), ℓ corresponds to the width of the routing space between the two logical patches. All logical error-rate polynomials in Eqs. (4)–(6) provide error rates per syndrome-measurement round. Per-round error rates are computed by varying the number of syndrome-measurement rounds for fixed d_x and d_z distances, repeating such procedures for different d_x and d_z distances, and fitting all the obtained data to an ansatz. Note that the d_z distance in Eq. (6) is taken to be the d_z distance of the full merged surface-code patches, whereas d_z in Eqs. (4) and (5) is the d_z distance of the individual logical patches (since logical Z errors are much less likely to occur when surface-code patches are merged, due to the increased d_z distance). In Fig. 9, we compare the best-fit polynomial $\mathbb{P}_{(0,1,0,0)}$ with a representative subset of our data obtained from our Monte Carlo simulations for various values of d_m, where the chosen parameters are described in the caption. The plot shows the exponential suppression in purely timelike error probabilities as a function of d_m and that the data are in good agreement with our best-fit polynomials. In Sec. VI, after introducing our protocol for minimizing routing costs, we use the logical failure-rate polynomials in Eqs. (3)–(6) to estimate the overhead costs for implementing quantum algorithms.

We conclude this section by pointing out that the above labels used in the logical error-rate polynomials, which represent different failure mechanisms that can occur during lattice surgery, can be generalized using $k + 2$ bits for an arbitrary $P_1 \otimes P_2 \otimes \cdots \otimes P_k$ Pauli measurement. Out of the $k + 2$ bits, k bits are used to represent a logical Pauli error on the logical patches that can also flip the parity of the measured Pauli (such as logical Z errors for $X \otimes X$). Another bit represents a logical parity measurement failure. The last bit encodes the logical Pauli error that affects all logical qubits in the merged patch (such as a logical X error during an $X \otimes X$ measurement).

IV. PROTOCOL FOR TWIST-FREE LATTICE SURGERY

Lattice surgery provides a fault-tolerant way to measure Pauli operators and is well suited for topological codes. However, not all Pauli operators are equally easy to measure. We say that an operator is a *XZ* Pauli when it is a tensor product of $\{\mathbb{1}, X, Z\}$. For *XZ* Pauli operators, standard lattice surgery suffices and a surface-code architecture would need only weight-4 stabilizer measurements. However, for some topological codes such the surface code, measuring Pauli operators containing any Y terms is more difficult. It has been shown that this can be achieved by introducing a twist defect for each Y in the Pauli operator [11,12,51]. For examples of twist-defect lattice surgery, see Fig. 1 of this work and Fig. 40(d) of Ref. [12]. However, each surface-code twist defect requires a stabilizer measurement on five physical qubits. This can be very challenging to implement in a 2D architecture with limited connectivity and could require multiple ancilla qubits. The additional ancilla qubits and gates will thus increase the total measurement-failure probabilities for weight-5 checks. Furthermore, even a single isolated weight-5 check will have an impact on the gate scheduling over the whole surface-code patch, which can introduce additional types of correlated errors. Lastly, twist-based surface codes coupled with a MWPM decoder have been shown to have a reduced effective code distance [52]. As such, we expect that twist-based Pauli measurements will suffer a performance loss relative to twist-free Pauli measurements. Any increases in measurement error probabilities during twist-based lattice surgery can be suppressed by extending d_m. In other words, we expect use of twists to increase the run time of lattice-surgery computations. The exact magnitude of this run-time cost is currently unknown and will depend on the precise twist implementation details and the noise model [53].

Here, we outline an alternative twist-free approach to measuring operators containing Pauli Y terms. The additional cost of supporting Pauli Y terms relative to measuring *XZ* Pauli operators is roughly a 2× slowdown in algorithm run time and a $+2$ additive cost in the number of logical qubits (although we show later that one of these logical qubits can be borrowed from space allocated to routing). Whether this 2× slowdown is preferable to the slowdown incurred by using twists is an open question, due to a lack of data on twist performance.

To explain our protocol, we make use of the notation

$$X[\mathbf{u}] := \prod_{j=1}^{N} X_j^{u_j}, \tag{7}$$

$$Z[\mathbf{v}] := \prod_{j=1}^{N} Z_j^{v_j}, \tag{8}$$

for any binary vectors $\mathbf{u} = (u_1, u_2, \ldots u_N)$ and $\mathbf{v} = (v_1, v_2, \ldots v_N)$. It is well known that any Hermitian Pauli operator can (up to a ± 1 phase) be decomposed as

$$P = i^{\mathbf{u}\cdot\mathbf{v}} Z[\mathbf{v}]X[\mathbf{u}]. \tag{9}$$

Then, $\mathbf{u}\cdot\mathbf{v} = \sum_j u_j v_j$ counts the number of locations where $X[\mathbf{u}]$ and $Z[\mathbf{v}]$ have overlapping support. Therefore, using $X_j Z_j \propto Y_j$, we see that $\mathbf{u}\cdot\mathbf{v}$ gives the number of Y terms in P. Furthermore, $X[\mathbf{u}]$ and $Z[\mathbf{v}]$ commute whenever $\mathbf{u}\cdot\mathbf{v}$ is even. Equivalently, whenever P contains an even number of Y terms, it can be decomposed (up to a phase) into a product of two commuting operators $X[\mathbf{u}]$ and $Z[\mathbf{v}]$.

Let us assume for now that $\mathbf{u}\cdot\mathbf{v}$ is even, returning to the odd case later. This suggests that we could measure P by using twist-free lattice surgery to measure $X[\mathbf{u}]$ and $Z[\mathbf{v}]$. However, this would reveal additional unwanted information about $X[\mathbf{u}]$ and $Z[\mathbf{v}]$. To obfuscate this unwanted information, we perform the protocol as illustrated in Fig. 10 and described below:

1. Prepare an ancilla (qubit A) in the state $|0\rangle$.
2. Measure $X[\mathbf{u}] \otimes X_A$ with outcome $m_x \in \{0, 1\}$.
3. Measure $Z[\mathbf{v}] \otimes X_A$ with outcome $m_z \in \{0, 1\}$.
4. Return $m_x \oplus m_z \oplus c$ as the outcome of $P = i^{\mathbf{u}\cdot\mathbf{v}} X[\mathbf{u}]Z[\mathbf{v}]$.
5. Measure qubit A in the Z basis with outcome $q \in \{0, 1\}$.
6. If $q = 1$, then apply a $Z[\mathbf{v}]$ correction (to the Pauli frame).

In step (4), we use a constant c that we define as follows:

$$c = \begin{cases} 0, & \text{if } \mathbf{u}\cdot\mathbf{v} = 0 \pmod 4, \\ 1, & \text{if } \mathbf{u}\cdot\mathbf{v} = 1 \pmod 4, \\ 1, & \text{if } \mathbf{u}\cdot\mathbf{v} = 2 \pmod 4, \\ 0, & \text{if } \mathbf{u}\cdot\mathbf{v} = 3 \pmod 4. \end{cases} \tag{10}$$

Clearly, the product of measurement outcomes in steps (2) and (3) gives $X[\mathbf{u}]Z[\mathbf{v}]$ up to some constant. However, at no point do we learn the value $Z[\mathbf{v}]$ or $X[\mathbf{u}]$; therefore the protocol works as claimed. In Appendix A, we provide a more formal proof of the correctness of our protocol and the derivation of the constant c.

We have assumed earlier that the Pauli operator P contains an even number of Y terms. To handle odd numbers of Y terms, we prepare an additional ancilla in the Y basis eigenstate $|Y\rangle = (|0\rangle + i|1\rangle)/\sqrt{2}$. Then, by measuring $Y \otimes P$, we effectively measure P. Furthermore, if P contains an odd number of Y terms, then $Y \otimes P$ contains an even number and can be measured using the above construction. This modified variant of the twist-free lattice-surgery measurement is also illustrated in Fig. 10. Note that the $Y \otimes P$ measurement does not affect the $|Y\rangle$ state

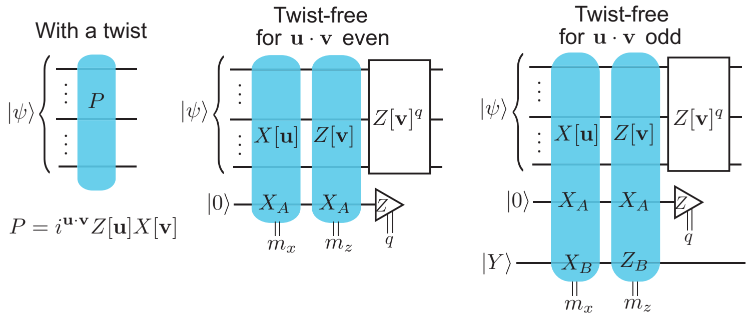

FIG. 10. Implementations of a Pauli measurement of $P = i^{\mathbf{u}\cdot\mathbf{v}}X[\mathbf{u}]Z[\mathbf{v}]$. Whenever P contains any Y terms, simple lattice-surgery operations cannot be used. The standard solution is to use twist-based lattice surgery. However, we show that the same outcome can be achieved twist free, with an extra $|Y\rangle$ ancilla used to handle cases where $X[\mathbf{u}]$ and $Z[\mathbf{v}]$ do not commute. The twist-free approaches report $m_x \oplus m_z \oplus c$ as the outcome for the measurement of P, where c is a constant determined by $\mathbf{u}$ and $\mathbf{v}$.

and so it can be reused many times and its preparation cost (e.g., through state distillation [7]) only needs to be paid once per algorithm and is therefore negligible.

Our twist-free approach uses up to two logical ancillas, a $|0\rangle$ ancilla that is repeatedly reset and sometimes a $|Y\rangle$ ancilla that is reused. Therefore, we have an additive +2 logical qubit cost. The run-time cost is dominated by steps (2) and (3). All other steps use only single-qubit operations that effectively take zero time in lattice surgery. Therefore, the run time doubles compared to the run time of measuring a Pauli operator free from Y terms. If steps (2) and (3) each fail with probability $\mathbb{P}$ [e.g., as in Eq. (1)], then the whole protocol fails with probability $\mathbb{P}' = 2\mathbb{P}(1-\mathbb{P}) \approx 2\mathbb{P}$. This is a minor effect, since failure probabilities are

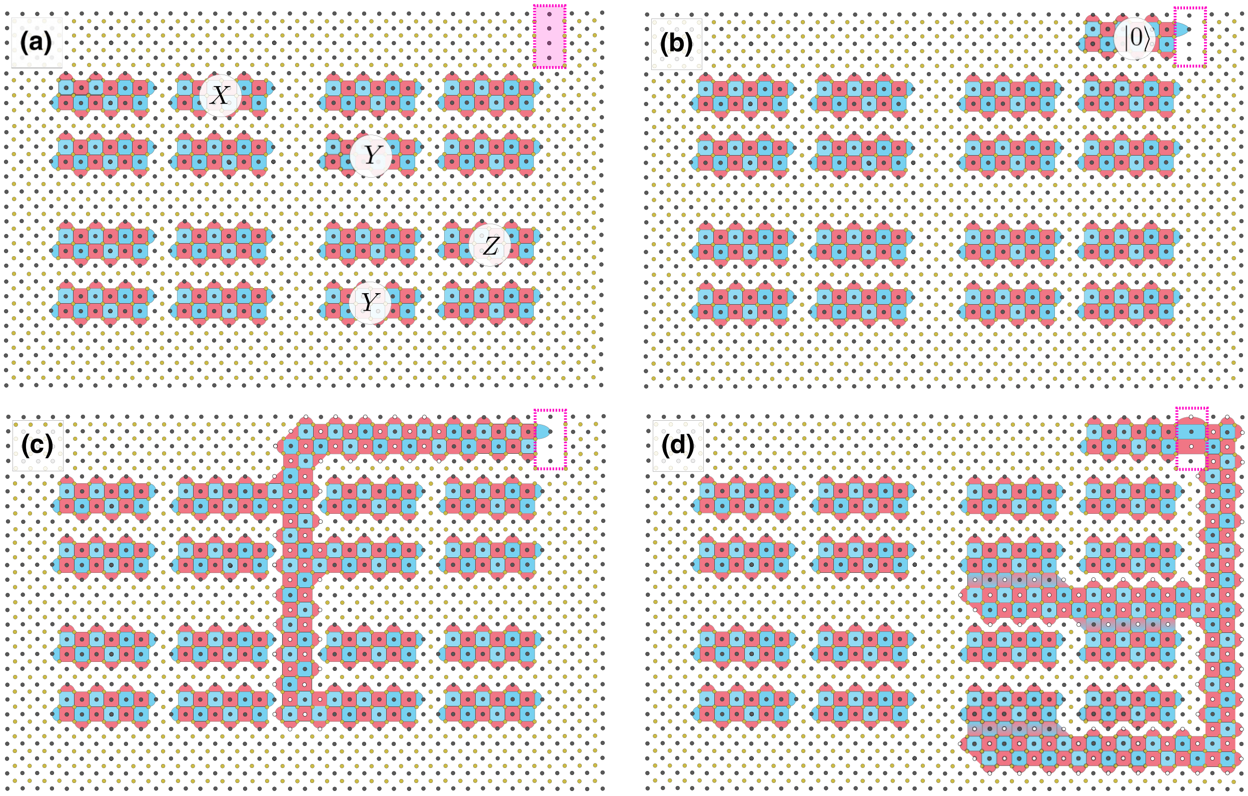

FIG. 11. An example of a 2D layout for a quantum computer and the implementation of twist-free lattice-surgery operations to realize a $P = Y \otimes Y \otimes X \otimes Z$ Pauli measurement, for which the circuit diagram is given in Fig. 10. (a) An initial layout of thin rectangular surface codes ($d_x = 3$ and $d_z = 7$), with labels showing the Pauli operators to be measured. Note the pink rectangle where the hardware layout is slightly adapted to enable elongated stabilizer measurements just within this region. The space between surface-code patches is referred to as the *routing space* and is used in the subsequent steps. (b) The preparation of a logical $|0\rangle$ state in the routing space. (c) A lattice-surgery measurement of $X \otimes X \otimes X \otimes \mathbb{1} \otimes X$. (d) A lattice-surgery measurement of $Z \otimes Z \otimes \mathbb{1} \otimes Z \otimes X$, where at every Z logical boundary we use a domain wall and at the single X logical boundary we use elongated stabilizers within the pink region. In both steps (c) and (d), the parity of the logical Pauli measurement is determined by the product of the stabilizers marked with a white vertex, with corrections applied by the decoder.

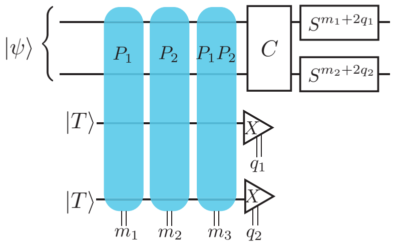

FIG. 12. A simple TELS protocol: it is an error-detecting version of the PBC used in Fig. 2, using the measurement code given in Eq. (14). While this approach uses three multiqubit Pauli measurements, the capability to detect an error means that lattice surgery can be executed over a shorter time d_m. If there is no error, we have that the measurement outcomes obey $m_1 \oplus m_2 = m_3$. When an error is detected, we simply remeasure the Pauli operators.

exponentially suppressed by increasing the run time d_m and using large enough d_x and d_z code distances.

In Fig. 11, we show an example of how the circuit picture of Pauli measurements in Fig. 10 can be explicitly mapped into a 2D lattice-surgery protocol consisting only of XZ-Pauli-operator measurements. We present our protocol for thin rectangular surface codes, although our protocol would also work for square surface-code patches. First, we note how the temporary $|0\rangle$ ancilla is prepared in the spare routing space provided for performing lattice surgery, so that it does not actually contribute to the space overhead. Note also that to accomplish the $Z[\mathbf{v}] \otimes X_A$ measurement in Fig. 11(d), there is one region (highlighted in pink) where we measure elongated stabilizers. Here, we assume that the hardware is permanently deformed in this region. In other words, the circuit is *hardwired* at this location, so that the elongated stabilizer can be measured with minimal performance loss compared to any other weight-4 stabilizers (e.g., by using longer resonators) and thus does not require additional ancilla qubits. Alternatively, these elongated stabilizers could also be measured in a homogeneous hardware layout but with a modified procedure for performing the measurement. For instance, one could use two ancilla qubits prepared in a Greenberger-Horne-Zeilinger (GHZ) state to measure the elongated stabilizers. However, since the result of the stabilizer measurement would be given by the product of the measurement outcomes of both ancillas, and due to the extra fault locations, the use of GHZ states would increase the total measurement-failure probability of the elongated checks. Another possibility would be to use the second ancilla qubit as a flag qubit [20,21,48,49,54–61]. However, by doing so, one might require an additional time CNOT step per round of stabilizer measurements to perform all two-qubit gates for the stabilizer measurements while avoiding scheduling conflicts.

V. TEMPORAL ENCODING FOR FAST LATTICE SURGERY

In Secs. II A and II B, we discuss the standard approach to PBC using lattice surgery and related algorithm runtime bottlenecks. In this section, we show how to exceed this bottleneck and run algorithms at faster clock speeds using TELS. The key idea behind TELS is to use fast noisy lattice-surgery operations, with this noise corrected by encoding the sequence of Pauli measurements within a classical error-correcting code. This encoding can be thought of as taking place in the time domain, so the encoding does not directly lead to additional qubit overhead costs. However, there can be a small additive qubit cost when TELS is used for magic state injection, with magic states needing to be stored for slightly longer times.

A. Parallelizable Pauli measurements

Here, we review parallelization, where the sequence of Pauli measurements can be grouped into sets of *parallelizable* Pauli measurements. Let $P_{[t,t+k]} := \{P_t, P_{t+1}, \ldots, P_{t+k}\}$ be a subsequence of our Pauli operators. We say that $P_{[t,t+k]}$ is a parallelizable set if they all commute and if any Clifford corrections can be commuted to the end of the subsequence. For example, we obtain a parallelizable set whenever we use magic states to perform a $T^{\otimes k}$ gate. In Fig. 2, we show several ways to implement $T^{\otimes 2}$ with the PBC approach, requiring two parallelizable measurements $\{P_1, P_2\} = \{CZ_1Z_3C^{\dagger}, CZ_2Z_4C^{\dagger}\}$. Therefore, given a circuit with μ T gates and T-depth γ, the Pauli-measurement sequence can always be split into a sequence of γ parallelizable sets of average size $k := \mu/\gamma$.

Fowler introduced the notion of time-optimal quantum computation [62] and Litinski (see Sec. 5.1 of Ref. [12]) showed how this can be realized using lattice surgery in a 2D layout. In time-optimal PBC, an n-qubit computation of T depth γ can be reduced to run time $O(n + \gamma)$. However, the space-time volume is never compressed by using the time-optimal approach, so that reducing the algorithm run time to 10% of a seqPBC run time would require at least a 10× increase in qubit cost. Litinski worked through some highly paralleziable examples in greater detail, showing that a reduction to 56.5% of seqPBC run time would need over 6× the qubit costs and that a reduction to 11% of seqPBC run time would need over 20× the qubit cost. The qualifier "over" in these estimates reflects that an increase in space-time volume also increases the code distance needed, further increasing the overhead of the time-optimal approach by a polylogarthmic factor on top of Litiniski's estimates. While this is a powerful approach to exploring space-time trade-offs, early fault-tolerant quantum computers will be qubit limited. Kim *et al.* [63] has also proposed another way to exploit large parallelizable sets, but they have used long-range gates that are not possible in 2D hardware and they have also made some strong assumptions regarding the speed at which transversal gates can be fault-tolerantly applied.

From the above, we see that it is crucial to understand the extent to which algorithms possess potential for parallelization. Fortunately, Kim *et al.* [63] have already studied quantum algorithms for chemistry and found k to vary between 9 and 14 depending on the orbitals used, so we regard this as a practically reasonable range.

B. Encodings and code-parameter proofs

Here, we introduce our own approach to exploiting parallelizable Pauli sets. Unlike previous time-optimal approaches, it does not incur a multiplicative qubit overhead cost and can reduce the overall space-time cost.

Due to the properties of a parallelizable Pauli set, all Pauli operators within the set can be measured in any order. Furthermore, we can measure any set $\mathcal{S}$ that generates the group $\langle P_t, P_{t+1}, \ldots, P_{t+k}\rangle$. If the set $\mathcal{S}$ is overcomplete, there will be some linear dependencies between the measurements that can be used to detect (and correct) for any errors in the lattice-surgery measurements. For example, consider the simplest parallelizable set $\{P_1, P_2\}$ as in Fig. 2 and let d_m be the required lattice-surgery time, so that performing both measurements takes $2(d_m+1)$ error-correction cycles. We could instead measure $\{P_1, P_2, P_1P_2\}$. If the third measurement outcome is not equal to the product of the first two measurements, then we know that something has gone wrong and we can repeat the measurements to gain more certainty of the true values. By measuring the overcomplete set $\{P_1, P_2, P_1P_2\}$, we have performed an extra lattice-surgery measurement but we have gained that we can tolerate a single lattice-surgery failure. This means that we could instead use $d'_m \ll d_m$ and still achieve the same overall success probability. If $3d'_m \ll 2d_m$, then the computation has been accelerated. This is the key insight behind TELS and next we dive deeper into more general encoding schemes and their performance.

In general, given a parallelizable Pauli set

$$\mathcal{P} = \{P_t, P_{t+1}, \ldots, P_{t+k-1}\}, \tag{11}$$

we can define operators generated from this set as follows:

$$Q[\mathbf{x}] := \prod_{j=0}^{k-1} P_{t+j}^{x_j}, \tag{12}$$

where $\mathbf{x}$ is a length-k binary column vector. Given a set that generates all the required Pauli operators, so that $\langle \mathcal{S}\rangle = \langle \mathcal{P}\rangle$, we can write the elements as

$$\mathcal{S} = \{Q[\mathbf{x}^1], Q[\mathbf{x}^2], \ldots, Q[\mathbf{x}^n]\}, \tag{13}$$

with the superscripts denoting different vectors. Since this is a generating set, the vectors $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n\}$ must span the relevant space. Furthermore, we can define a matrix G with these vectors as columns and this matrix specifies the TELS protocol. In the simple $k=2$ example where $\mathcal{S} = \{P_1, P_2, P_1P_2\}$, we would have that

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x}^1 & \mathbf{x}^2 & \mathbf{x}^3 \end{pmatrix}. \tag{14}$$

Note that the rows of this matrix generate the code words of the $[3,2,2]$ classical code. In general, we consider G as the generator matrix for the code words of an $[n,k,d]$ classical code and we call this the *measurement code* for the protocol. Note that k is the number of (unencoded) Pauli operators in the generating set. We only consider full-rank G, where k equals the number of rows in G. The number n represents how many Pauli measurements we physically perform in the encoded scheme and corresponds to the number of columns in G. The distance d is the lowest-weight vector in the row span of G.

Next, we show that the code distance d does indeed quantify the ability of TELS to correct errors. First, we formalize the redundancy in the set of lattice-surgery measurements. For any length-n binary vector $\mathbf{u} = (u_1, u_2, \ldots, u_n)$, we have that

$$\prod_{j:u_j=1} Q[\mathbf{x}^j] = Q\left[\sum_l u_l \mathbf{x}^l\right]. \tag{15}$$

Since the matrix G is full rank and has more columns than rows, there will exist $\mathbf{u}$ such that $\sum_j u_j \mathbf{x}^j = 0$. For these $\mathbf{u}$, we have that

$$\prod_{j:u_j=1} Q[\mathbf{x}^j] = \mathbb{1}. \tag{16}$$

Therefore, these $\mathbf{u}$ vectors describe redundancy in the measurements. The condition $\sum_j u_j \mathbf{x}^j = 0$ can be rewritten compactly as $G\mathbf{u} = 0$. Following the convention in coding theory, this set of $\mathbf{u}$ is called the dual of G and denoted

$$G^\perp := \{\mathbf{u} : G\mathbf{u} = 0 \pmod 2\}. \tag{17}$$

Next, we consider how this redundancy is used to detect timelike lattice-surgery errors. We let $\mathbf{m} = \{m_1, m_2, \ldots m_n\}$ be a binary vector denoting the outcomes of the lattice-surgery Pauli measurements in the set $\mathcal{S}$. That is, if a measurement of $Q[\mathbf{x}^j]$ gives outcome "+1" we set $m_j = 0$ and when the measurement of $Q[\mathbf{x}^j]$ gives "-1" we set $m_j = 1$. Given a $\mathbf{u} \in G^\perp$, we know the Pauli operators product to the identity [recall Eq. (16)], so when there are no timelike lattice-surgery errors, we have

$$\prod_{j:u_j=1} m_j = \mathbf{u}\cdot\mathbf{m} = 0 \pmod 2. \tag{18}$$

Conversely, if we observe

$$\prod_{j:u_j=1} m_j = \mathbf{u}\cdot\mathbf{m} = 1 \pmod 2, \tag{19}$$

then we know a timelike lattice-surgery error must have occurred. Let us write $\mathbf{m} = \mathbf{s} + \mathbf{e}$, where $\mathbf{s}$ is the ideal measurement outcome and $\mathbf{e}$ is the measurement error. The ideal measurement outcomes are self-consistent and so always satisfy $\mathbf{u}\cdot\mathbf{s} = 0$ for all $\mathbf{u} \in G^{\perp}$. Therefore, we see that an error $\mathbf{e}$ is undetected if and only if $\mathbf{u}\cdot\mathbf{e} = 0$ for some $\mathbf{u} \in G^{\perp}$. This is equivalent to undetected errors $\mathbf{e}$ being in the row span of G (since the dual of the dual is always the original space). Recall that the distance d denotes the lowest- (nonzero) weight vector in the row span of G. Therefore, d also denotes the smallest number of timelike lattice-surgery errors needed for them to be undetected by TELS. Consequently, if $\mathbb{P}$ is the probability of a single timelike error, TELS error detection will fail with probability $O(\mathbb{P}^d)$.

Matrices such as G also appear in the literature in the context of measuring overcomplete sets of stabilizers for some quantum error-correction code. In the error-correction setting, these codes have been called measurement codes [64], metachecks [65,66], symmetries [67], and syndrome-measurement codes [68]. However, we deploy this idea in the context of lattice surgery as a strategy to improve algorithm run times.

C. Examples and numerics

The simplest examples of TELS will detect a single error. Given a Pauli set $\{P_t, P_{t+1}, \ldots, P_{t+k}\}$, we measure each of these observables separately and then their product, so that the measurement code has generator matrix

$$G = \begin{pmatrix} 1 & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 1 \end{pmatrix}, \tag{20}$$

which is an identity matrix padded with an extra column that is an all-1 vector. Therefore, this corresponds to a $[\alpha+1, \alpha, 2]$ classical code that detects a single error. Concatenation of such a code m times gives a code with parameters $[(\alpha+1)^m, \alpha^m, 2^m]$.

We can also consider using a simple $[8,4,4]$ extended Hamming code as the measurement code, with generator matrix

$$G = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{21}$$

This corresponds to replacing $\{P_1, P_2, P_3, P_4\}$ with $\mathcal{S}$ containing the eight operators

$$\mathcal{S} = \{P_2P_3P_4, P_2P_3, P_2P_4, P_2, P_1P_3P_4, P_1P_3, P_1P_4, P_1\}. \tag{22}$$

Because the generator matrix has distance 4, this scheme will detect up to three errors. This Hamming code is the $m = 3$ member of a family of $[2^m, 2^m - m - 1, 4]$ extended Hamming codes.

There are several viable strategies to handle a detected error. Here, we consider the following detect-and-remeasure strategy: if a distance-d measurement code is used with lattice surgery performed for time d_m, then whenever an error is detected we “remeasure,” but this time using the original Pauli set $\mathcal{P}$ instead of using the overcomplete set $\mathcal{S}$. For the remeasure round, we perform lattice surgery using an amount of time $\lceil qd_m \rceil$, where q is some constant scaling factor the value of which we discuss shortly. The expected run time to execute the protocol is then

$$T = n(d_m + 1) + p_d kqd_m, \tag{23}$$

where p_d is the probability of detecting an error. When we do not detect an error, the probability of a failure is $O(p^{d(d_m+1/2)}) \approx O(p^{dd_m/2})$. When we do detect an error, the remeasure round will fail with probability $O(p^{(qd_m+1)/2}) \approx O(p^{qd_m/2})$. The total failure probability will then be $O(p^{dd_m/2} + p_d p^{qd_m/2})$. Therefore, we can ensure that the total failure probability is $O(p^{dd_m/2})$ by setting $q = d$. However, due to constant factors, the optimal choice of q may be slightly different from $q = d$ and so we numerically optimize from this initial guess. When an error detection occurs, this leads to a long delay of time kqd_m to implement the remeasure round, but in practice p_d is so small that this has minimal impact on the expected run time.

We could alternatively just measure the overcomplete set $\mathcal{S}$ and run the measurement code in error-correction mode with lattice surgery repeated for time d_m'. Then, the protocol would fail with probability approximately $O(p^{dd_m'/4})$. Compared to the detect-and-remeasure strategy, we need $d_m' \approx 2d_m$ to achieve the same failure probability.

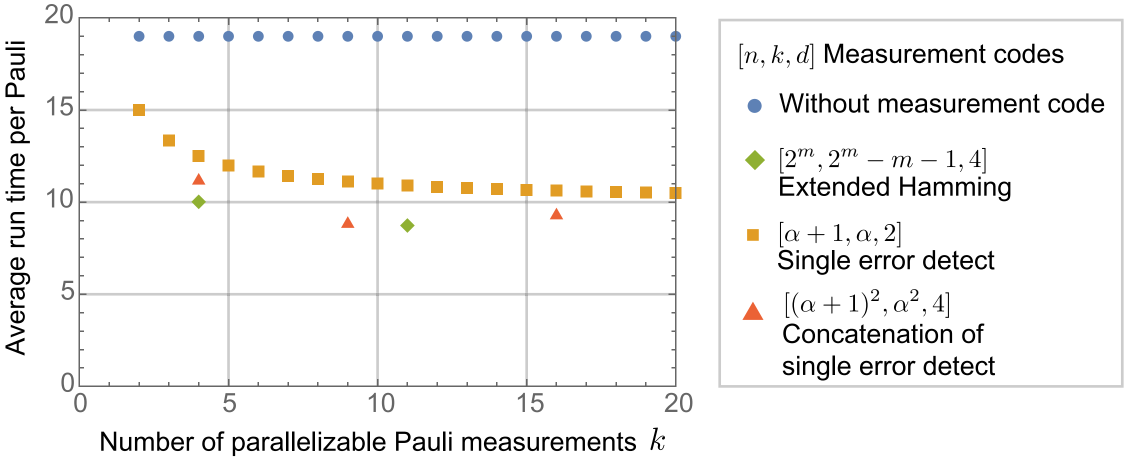

FIG. 13. A comparison of the standard lattice-surgery approach (without the measurement code) with three different TELS schemes for performing a set of k parallelizable Pauli measurements. We assume that the Pauli measurements give an incorrect outcome due to a timelike failure with a probability determined by Eq. (3) with $d_x\ell = 100$ and $p = 10^{-3}$. We set the allowed error per Pauli at $\delta = 10^{-15}$. The $[2^m, 2^m - m - 1, 4]$ are well-known Hamming codes. The $[\alpha + 1, \alpha, 2]$ are single error-detection codes and $[(\alpha + 1)^2, \alpha^2, 4]$ are concatenated single error-detection codes. While there are big jumps in k between the best-performing codes, these jumps could be partially smoothed out by considering other codes such as concatenated codes with different inner and outer code sizes, such as the $[(\alpha + 1)(\beta + 1), \alpha\beta, 4]$ codes. We also consider the triply concatenated codes with parameters $[(\alpha + 1)^3, \alpha^3, 8]$ but they perform poorly in the parameter regime shown here and so are omitted for clarity.

The run time of an error-correction scheme is then

$$T' = n(d'_m + 1) \approx n(2d_m + 1). \tag{24}$$

Compared to Eq. (23), in T' we drop the second term at the price of roughly doubling the first term. However, the second term is small because p_d is small, so overall error correction is not favorable compared to our detect-and-remeasure scheme.

Figure 13 shows some example numerical results using distance-2 and -4 codes. For example, when performing $k = 11$ parallelizable gates, the TELS scheme using extended Hamming codes will have a run time of 46% that of a standard seqPBC approach that measures the original parallelizable Pauli set $\mathcal{P}$. Since Kim *et al.* [63] have found that interesting quantum algorithms can have average k between 9 and 14, this suggests around a 2.2× speed-up due to TELS on practical problems. To obtain a similar speed-up, Litinski has estimated that the time-optimal approach would cost over 6× in qubit overhead. The TELS scheme has no multiplicative qubit overhead, although it does have a small additive qubit overhead, as all k magic states must be stored for the full duration of the protocol. However, fault-tolerant algorithms typically have $N \gg 100$ logical qubits and so the increase in logical qubits $N \rightarrow N + k$ is small. Indeed, overall the space-time volume will decrease, which is impossible using Fowler's time-optimal approach.

We do not find any examples of higher-distance codes ($d > 4$) that perform better in this parameter regime (e.g., $\delta = 10^{-15}$). Going to even lower error rates ($\delta \ll 10^{-15}$) or changing the noise model, then higher-distance codes become useful and the advantage improves further. Indeed, because of the existence of good classical codes with $n/k = O(1)$ and $d = \Omega(k)$, we know that TELS will asymptotically (for large k) be able to execute k parallelizable Pauli measurements in $O(k)$ time and with error $\delta = O(p^{\alpha k})$ for some constant α. In contrast, a standard seqPBC with unencoded lattice surgery would take run time $O(k\text{polylog}(k))$ to achieve the same error.

VI. THE CORE-CACHE ARCHITECTURE AND ROUTING OVERHEADS

In this section, we discuss a layout and data-access structure for a quantum computer that extends on the layout given in Fig. 11(a). We consider patches of (possibly rectangular) surface codes of size d_z by d_x. Between these patches we have some qubits dedicated as a lattice-surgery “bus” or routing space. We say that the routing space supports fast access if logical X and Z operators of every patch are adjacent to the routing space. Litinski has proposed several data-access structures [12], with his fast data structures using two-tile two-qubit patches (surface-code patches that each encode two logical qubits) that are sometimes called hexon surface codes.

In Sec. VI A, we show that the hexon approach is not necessary and give a layout for a quantum core (what Litiniski calls a fast data-access structure). Furthermore, we show that a lower routing overhead is possible when the surface-code patches are thin rectangles (e.g., $d_z \gg d_x$), as is the optimal choice for highly biased noise. In Sec. VI B,

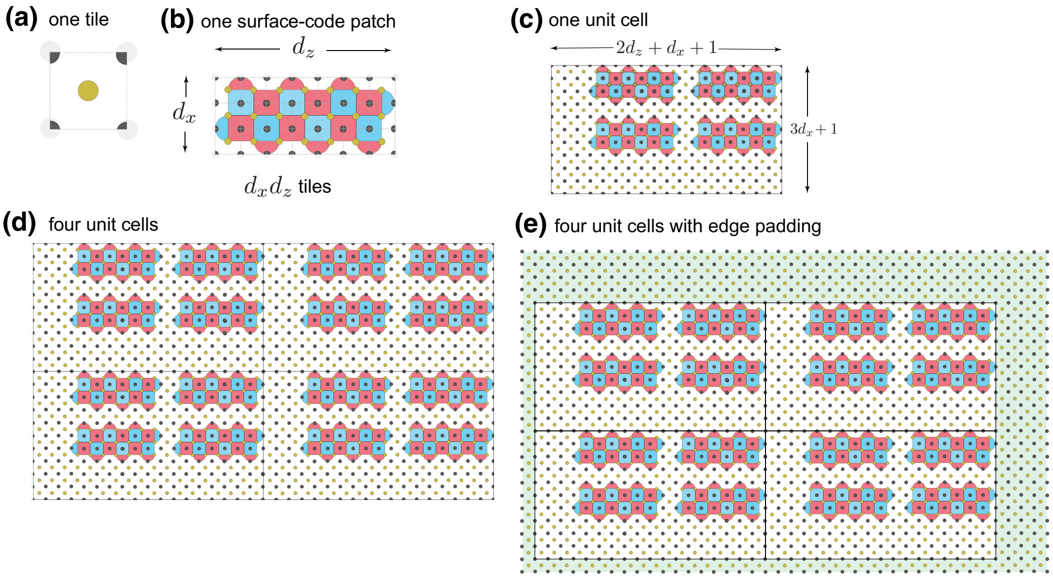

FIG. 14. The building blocks for a quantum computer. (a) A single tile containing two qubits: the ancilla qubits (cut at the four corners of the tile) are shared with neighboring tiles. (b) A single (asymmetric) surface-code patch. The number of tiles needed is the height times the width and so $d_x d_z$. (c) A single unit cell containing four surface-code patches and some routing space. Note that all surface-code patches have a X and Z logical boundary adjacent to either routing space within the bulk or the boundary of the unit cell. (d) Four unit cells tiled together. (e) The same as (d) but with the inclusion of extra padding highlighted in green at the edges to provide access to the remaining X and Z logical boundaries and ample routing space. There is more padding on the top and right since we need room to access the remaining X and Z logical operators.

we also discuss the idea of a core-cache model where logical qubits are temporally moved out of the fast data-access structure to reduce the routing overhead.

A. A quantum computer core

We count resource costs in terms of square tiles as defined in Fig. 14(a). Each tile contains a single data qubit and four quarter-ancilla qubits. Therefore, a device with T tiles will require roughly $2T$ qubits. However, we cannot cut qubits into quarters and so a precise counting will include these. For instance, a rectangular device with a height of h tiles and width of w tiles would have a total of $T = wh$ tiles and $2T + w + h + 1$ qubits. When the device is roughly square, then h and w are of size $O(\sqrt{T})$ and so there is a negligible additive cost compared to $2T$. We can realize a surface-code patch using $d_x d_z$ tiles as in Fig. 14(b) and therefore $2d_x d_z$ qubits. The number of data and ancilla qubits actively used in the surface-code patch is $2d_x d_z - 1$ and so when we try to pack them in a 2D arrangement, the tightest possible packing will contain one idling qubit per patch.

We collect surface-code patches into groups of four, which we call a unit cell [see Fig. 14(c)]. These unit cells are then repeated as shown in Fig. 14(d) to obtain the required number of logical qubits. Furthermore, we arrange the unit cells to form a quantum “core” and assume some additional padding shown in Fig. 14(e). Note that in Fig. 14, every patch has logical X and Z boundary operators connected to the routing space, which enables us to quickly perform multiqubit Pauli measurements between any subset of qubits within the core. Additionally, there are unused qubits between some of the surface-code patches. The spacing of the qubits ensures that lattice surgery can be performed (as we saw in Fig. 11) without using lattice twists that incur additional practical difficulties to implement in fixed- and low-connectivity hardware. In contrast, the data-access structures proposed by Litinski [12] have assumed liberal use of twists.

The routing overhead for unit cells is then the ratio of the number of tiles divided by the cost without any routing space (e.g., $4d_z d_x$):

$$O^{(\text{unit cell})}_{(d_z,d_x)} = \frac{(2d_z + d_x + 1)(3d_x + 1)}{4d_z d_x}. \tag{25}$$

The overhead for the entire core includes a contribution from the additional padding shown in Fig. 14(e). However, in the limit of many unit cells, the total overhead is dominated by the unit-cell overhead. In the limit of large distances $d_z, d_x \gg 1$, we have

$$O^{(\text{unit cell})}_{(d_z,d_x)} \approx \frac{3}{2} + \frac{3}{4}\frac{d_x}{d_z}. \tag{26}$$

Therefore, in the limit of large noise bias, $d_z \gg d_x$, the routing overhead factor is 1.5. We can compare this with

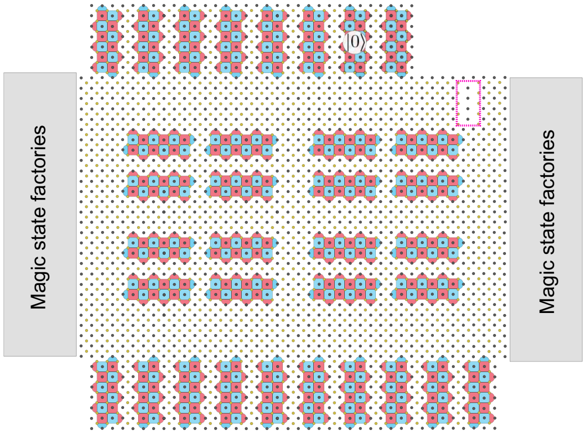

FIG. 15. A small example of a quantum computer layout, with a core composed of four unit cells (and therefore 16 surface-code patches); cache on the top or bottom edge of the core; and magic state factories on the left or right of the core. A pink dashed rectangle in the top-right corner indicates a hardwired lattice defect so as to enable twist-free lattice surgery.

Litinski's fast data blocks (see, e.g., Fig. 23 of Ref. [12]), where the overhead factor is 2 (in the large device limit) and so is more expensive. In contrast, for unbiased noise and $d_z = d_x$, our scheme has an asymptotic routing overhead factor of 9/4, which is slightly worse than Litinski's multiplicative factor-of-2 routing overhead. Indeed, solving Eq. (26) equal to 2 shows that $d_x < (2/3)d_z$ is the condition for our approach to have a routing overhead advantage. A more general analysis of the routing overhead that includes the green padded regions shown in Fig. 14(e) and contributions from the cache (see Sec. VI B) is given in Appendix C. We also point out that routing overhead is not the only important figure of merit. Our proposed design avoids twist defects and other significant lattice irregularities and therefore may be useful even without noise bias.

B. A quantum computer cache

We now proceed to build additional structure around the core. Using state distillation to prepare magic states, we need factories that supply the core. The purely fast data-access approach prioritizes speed over qubit cost. Here, we also discuss the idea of a core and cache architecture, where some logical qubits are temporarily stored in a quantum analog of cache. However, with some time cost, logical qubits can be quickly swapped in and out of the cache. A small-scale sketch of a device comprising core, cache, and magic state factories is illustrated in Fig. 15.

Packing qubits more compactly in the cache will clearly reduce the overhead costs. However, such a layout comes at a price, since only the X logical operators of these qubits can be accessed when it is in the cache. To access the logical Z operators, it must be swapped out of the cache and into the core. For a qubit stored in the cache, we can perform the following operations:

(1) Perform single-qubit X or Z measurements for a qubit in the cache (time cost zero).

(2) Measure multiqubit operators of the form $A \otimes B$, where B acts on the cache qubits and is a tensor product of X operators only and A acts on the core qubits and can be an arbitary Pauli operator (time cost d_m).

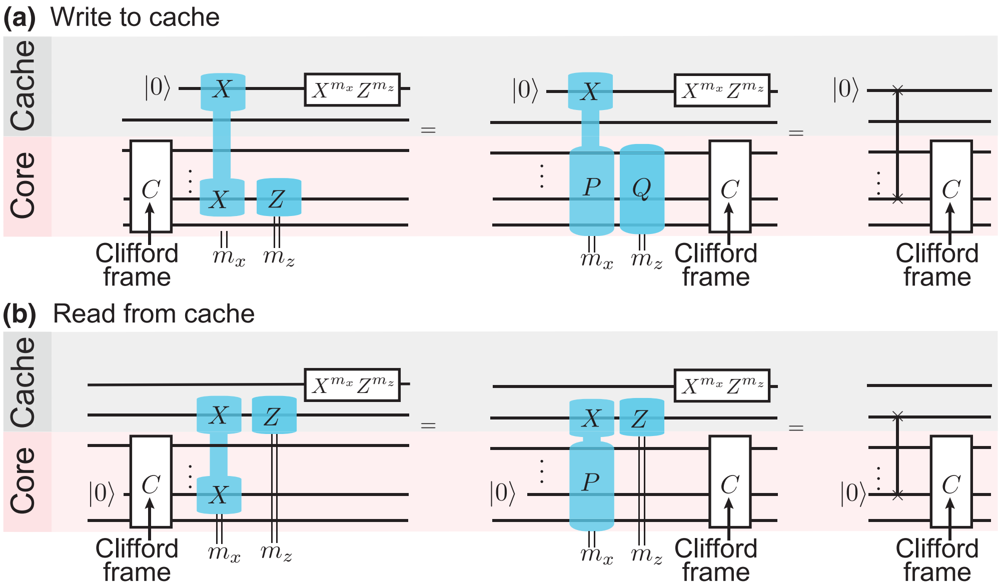

FIG. 16. Circuit diagrams for the operations (a) write to cache (WTC) and (b) to read from cache (RFC).

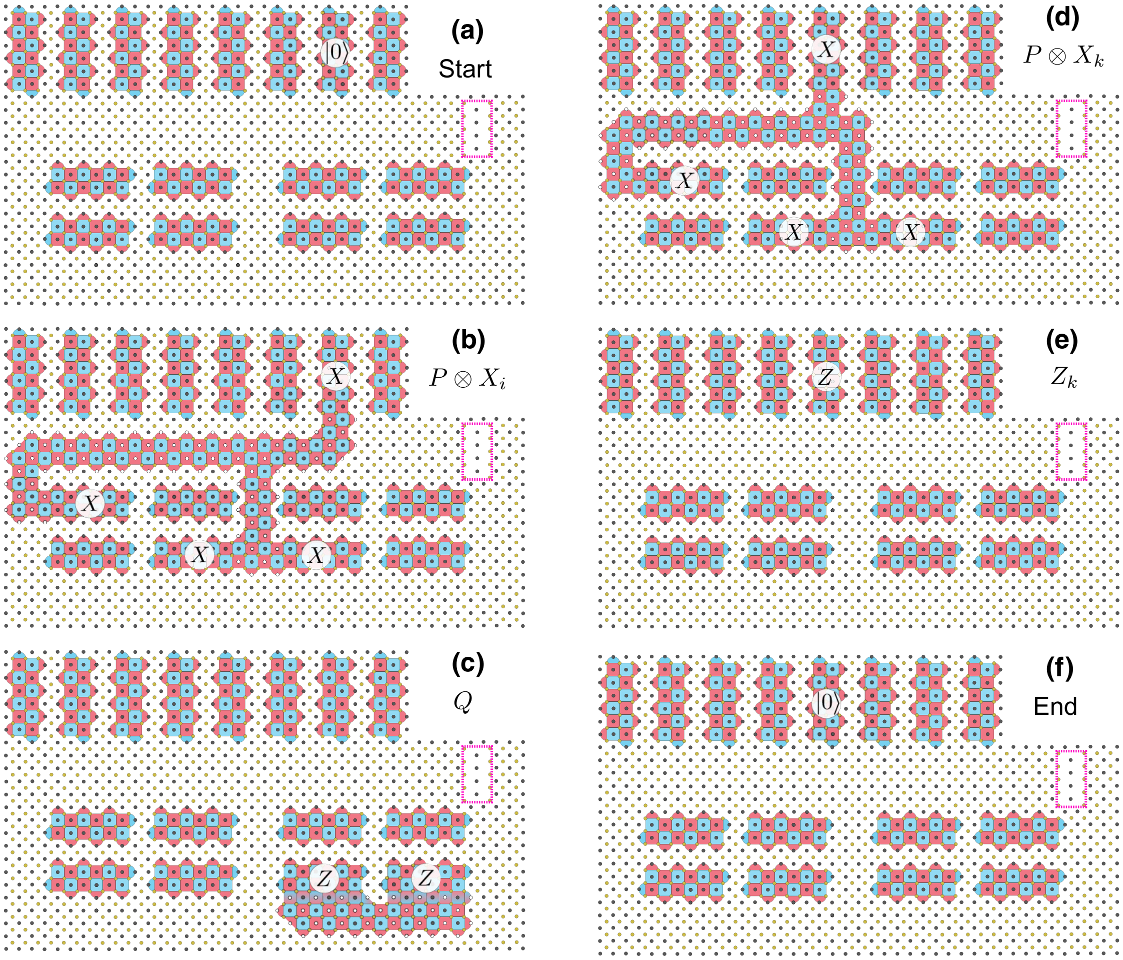

FIG. 17. Example lattice-surgery diagrams for using write to cache and read from cache to exchange qubits j and k between core and cache. (a) An illustration of the initial configuration with a $|0\rangle$ ancilla (index label i) in the cache. Qubit j is in the core and qubit k is in the cache and we wish to swap their locations. (b) We measure $P \otimes X_i$, where $P = CX_j C^\dagger$, in which C is the Clifford frame. For simplicity, we assume that P is composed of only X operators. (c) We measure $Q = CZ_j C^\dagger$ and this time assume that it is composed of only Z operators. (d) We measure $P \otimes X_k$, where $P = CX_j C^\dagger$ is the same operator as in (b). (e) We measure the single-qubit Pauli Z_k. If either of the simplifying assumptions we made for P or Q do not hold, then we need to use the twist-free protocol (or use twists). (f) At the end of the protocol, qubit j is in the cache, qubit k is in the core, and there is a $|0\rangle$ ancilla in the cache ready for further swaps.

(3) Perform Pauli updates to the Pauli frame (in software).
(4) Perform Cliffords to qubits in the core, by updating the Clifford frame (in software).

For algorithms where swapping in or out of the cache can be made infrequently (compared to other time costs), our approach reduces the routing overhead, with a mild impact on the algorithm run time. Note also that our core-cache architecture can be used in combination with the twist-free or TELS schemes already proposed.

This leaves the question of how to swap the location of qubits from the cache to the core. We cannot directly implement the Clifford SWAP operation, since Clifford operations can only be performed on core qubits. Furthermore, surface-code patches cannot be moved around to swap their positions, since such operations would require the Clifford frame C to be relabeled. Such a relabeling might make C act nontrivially on qubits in the cache. Rather, when performing a SWAP from a qubit in the core to the cache, we need to clean the Clifford frame so that it only acts on core qubits.

We now define two elementary operations—write to cache (WTC) and read from cache (RFC)—which, when combined, enable a Clifford-cleaned swap. We first present the WTC and RFC protocols using circuit diagrams in Fig. 16. They are both essentially two-qubit teleportation protocols but with the Clifford frame adapting the Pauli measurements performed. The WTC operation uses two multiqubit Pauli measurements, whereas RFC can be performed faster as it uses only one multiqubit Pauli measurement and a single-qubit Pauli measurement (that takes zero time). The WTC operation requires a logical $|0\rangle$ ancilla in the cache, which through the protocol swaps place with a logical qubit in the core. The RFC operation requires a logical $|0\rangle$ ancilla in the core, which through the protocol swaps place with a logical qubit in the cache.

For a pair of logical qubits, one in the core and one in the cache, we can cleanly SWAP their positions, by executing WTC followed by RFC. The $|0\rangle$ ancilla initially in the cache moves to the core, then back to the cache but to a different cache location compared to where it started. The whole swap procedure requires three multiqubit Pauli measurements. Figure 17 shows an example using lattice surgery. This figure shows a simple scenario where these three multiqubit Pauli measurements are XZ Pauli measurements (even after conjugated by the Clifford frame) and so can be realized with three simple lattice-surgery operations. However, more generally, when some measurements are not of XZ type, we need to either use twist defects (and benchmark their performance) or use our twist-free protocol and realize the swap with up to six lattice-surgery operations.

In Appendix C, we perform a resource-cost analysis for simulating the Hubbard model using the full core-cache architecture described in this section. In particular, we provide a rigorous analysis of routing overhead costs including contributions from the cache and green padded region in Fig. 14(e).

VII. CONCLUSION

In Sec. III, we introduce a decoding algorithm compatible with lattice-surgery protocols and numerically compute failure-rate polynomials for the dominant failure mechanisms of an $X \otimes X$ Pauli measurement. Our analysis allows one to compute appropriate d_x and d_z code distances, as well as the number of syndrome-measurement rounds d_m during lattice surgery for successfully implementing algorithms.

In Sec. IV, we introduce a twist free protocol for measuring arbitrary Pauli operators using lattice surgery. The protocol incurs a multiplicative factor-of-2 slowdown in algorithm run time. However, surface codes with twists require higher-weight-stabilizer measurements and increased gate-scheduling complexities and have a reduced effective code distance when using a MWPM decoder. Such features inevitably cause a reduction in performance compared to lattice-surgery protocols involving only X and Z Pauli measurements. Consequently, a careful numerical analysis with twists is needed in order to determine whether using twists can beat the 2× cost of the twist-free approach.

In Sec. V, we introduce a technique that we call temporal encoding of lattice surgery. By encoding lattice-surgery measurements in the time domain, we show that a 2× reduction in algorithm run times can be achieved for quantum algorithms of practical scale without incurring additional qubit overhead costs. For more highly parallelizable algorithms or larger algorithms, the use of larger classical code distances and the exploration of other code families will lead to even greater improvements in algorithm run times. Since posting a preprint of this work.

Lastly, in Sec. VI, we provide a core-cache architecture compatible with our lattice-surgery protocols. A subset of the data qubits are stored in a cache, which reduces the footprint of the routing space, and can be quickly accessed when Z measurements need to be performed. We find that for such an architecture, the routing of overhead costs adds a multiplicative factor-of-1.5 increase to the total resource costs for performing lattice surgery. A clear direction for future work would be to analyze such architectures in the presence of twists in order to better understand the trade-offs with using a twist-free lattice-surgery protocol.

Apart from considering twists, a direction of future work would be to apply our methods using other error-correcting codes to potentially achieve lower resource costs. Promising code candidates include codes tailored for biased noise such as the XZZX surface codes [67,69–71], subsystem codes with high thresholds [72], and other code families with high encoding rates, such as hyperbolic surface codes [73,74].

ACKNOWLEDGMENTS

We thank Giacomo Torlai for his help with using the AWS clusters. We thank Oscar Higgott, Fernando Brandao, and Noah Shutty for their comments and suggestions on our manuscript.

Craig Gidney noticed that temporal encoding of lattice surgery could be used to reduce the runtime of magic state distillation factories.

APPENDIX A: TWIST-FREE PROOF

Here, we give a formal proof that the twist-free lattice-surgery protocol works as claimed. Consider the case when step (5) yields a $q = 0$ outcome so that we project onto the $|0\rangle$ state. Then, steps (1)–(5) implement

$$M_{+1} := |0\rangle\langle 0|_A \Pi_{ZX}(m_z)\Pi_{XX}(m_x)|0\rangle\langle 0|_A, \qquad \text{(A1)}$$

where

$$\Pi_{ZX}(m_z) := \frac{1}{2}\left\{\mathbb{1} \otimes \mathbb{1} + (-1)^{m_z} Z[\mathbf{v}] \otimes X_A\right\}, \qquad \text{(A2)}$$

$$\Pi_{XX}(m_x) := \frac{1}{2}\left\{\mathbb{1} \otimes \mathbb{1} + (-1)^{m_x} X[\mathbf{u}] \otimes X_A\right\}. \qquad \text{(A3)}$$

Using that for arbitrary Q,

$$|0\rangle\langle 0|_A(Q\otimes \mathbb{1})|0\rangle\langle 0|_A = Q\otimes |0\rangle\langle 0|_A,$$
$$|0\rangle\langle 0|_A(Q\otimes X_A)|0\rangle\langle 0|_A = 0, \qquad \text{(A4)}$$

we deduce

$$M_{+1} = \frac{1}{4}\left\{\mathbb{1} + (-1)^{m_x+m_z}(Z[\mathbf{v}]X[\mathbf{u}])\right\}\otimes |0\rangle\langle 0|_A, \quad \text{(A5)}$$

which is proportional to the projector for a $Z[\mathbf{v}]X[\mathbf{u}]$ measurement with outcome $m_x \oplus m_z$. When $\mathbf{u}\cdot\mathbf{v} = 0 \pmod 4$, we have $P = Z[\mathbf{v}]X[\mathbf{u}]$ and so $m_x\oplus m_z$ is the outcome of measuring P (justifying $c = 0$ in this case). On the other hand, if $\mathbf{u}\cdot\mathbf{v} = 2 \pmod 4$, we have $P = -Z[\mathbf{v}]X[\mathbf{u}]$ and so $m_x\oplus m_z\oplus 1$ is the outcome of measuring P (justifying $c = 1$ in this case). Recall that we are currently assuming that $\mathbf{u}\cdot\mathbf{v}$ is even.

In the event that step (5) yields a $q = 1$ outcome, we have that

$$M_{-1} := |1\rangle\langle 1|_A \Pi_{ZX}(m_z)\Pi_{XX}(m_x)|0\rangle\langle 0|_A. \qquad \text{(A6)}$$

Using that for arbitrary Q,

$$|1\rangle\langle 1|(Q\otimes \mathbb{1})|0\rangle\langle 0| = 0,$$
$$|1\rangle\langle 1|(Q\otimes X_A)|0\rangle\langle 0| = Q\otimes |1\rangle\langle 0|, \qquad \text{(A7)}$$

we have

$$M_{-1} = \frac{1}{4}\left\{(-1)^{m_z}Z[\mathbf{v}] + (-1)^{m_x}X[\mathbf{u}]\right\}\otimes |1\rangle\langle 0|_A, \quad \text{(A8)}$$

$$= \left\{(-1)^{m_z}Z[\mathbf{v}]\right\}\otimes X_A M_{+1}. \qquad \text{(A9)}$$

Therefore, we see that M_{-1} differs from M_{+1} by a $Z[\mathbf{v}]$ correction that we perform in step (6). There is also a global phase $(-1)^{m_z}$ but this is unimportant.

We remark that for the case where $\mathbf{u}\cdot\mathbf{v} = 1 \pmod 4$, adding an additional Y operator to P and performing the measurement using the $|Y\rangle$ ancilla is identical to the case where $\mathbf{u}\cdot\mathbf{v} = 2 \pmod 4$; hence c is 1. A similar argument can be used to show that $c = 0$ for the case where $\mathbf{u}\cdot\mathbf{v} = 3 \pmod 4$.

APPENDIX B: TREATING SPACE-TIME CORRELATED EDGES INCIDENT TO PARITY VERTICES

As mentioned in Sec. III, the space-time correlated edges incident to the parity vertices during the first round of the merged surface-code patches (i.e., vertices $v \in V_{\text{par}}^{(r+1)}$) need to be treated with care. Such edges are highlighted in red, black, and purple in Figs. 18(b) and 18(c) for an $X\otimes X$ measurement.

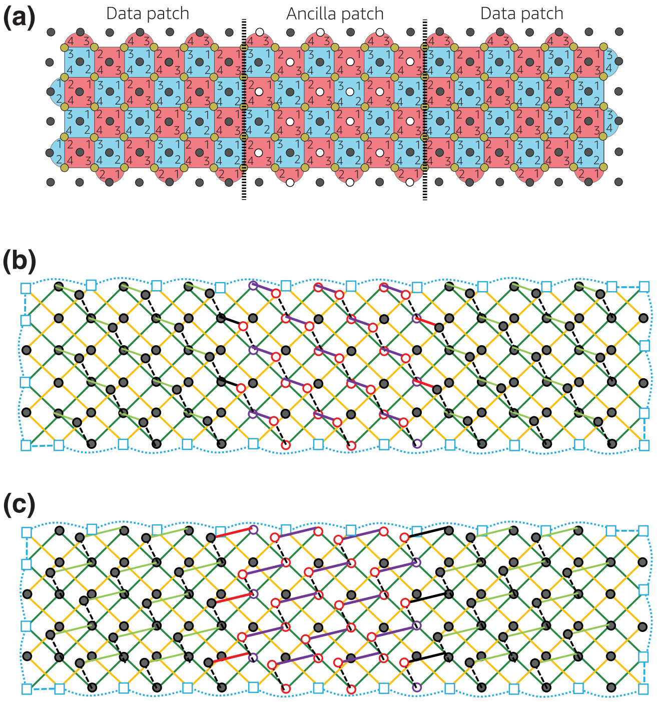

FIG. 18. (a) A surface-code lattice with $d_x = 5$, $d_z = 7$. The numbers in each stabilizer measurement plaquettes indicate the gate scheduling of the two-qubit gates (which in our simulations are CNOT gates). (b) The first type of space-time correlated edges for X stabilizers. (c) The second type of space-time correlated edges for X stabilizers. Space-time correlated edges incident to vertices present in the routing space are assigned different colors other than green for the reasons described in the text.

First, in round $r+1$, edges highlighted in purple must have infinite weight and can thus be removed. This is due to the fact that when the logical patches are merged using the routing space in round $r+1$, individual X-stabilizer measurements performed in the routing space (the ancillas of which are marked by white vertices) will have random outcomes and thus cannot be highlighted. As such, failures arising from two-qubit gates performed in the routing space, and which introduce errors that do not interact with stabilizers belonging to logical patches, cannot generate a nontrivial measurement outcome between two consecutive rounds of stabilizer measurements. Although the purple edges discussed above must be removed when they are incident to vertices in rounds $r+1$ and $r+2$, such edges incident to vertices v_i and v_j belonging to rounds greater than $r+1$ must be included since they will have finite weights.

Second, space-time edges incident to a single parity vertex $v \in V_{\text{par}}^{(r+1)}$ are highlighted in red in Figs. 18(b) and 18(c). Note that in round $r+1$, such edges are incident to parity vertices colored in purple, which are also transition vertices. Such edges arise from two-qubit gate failures in round $j \geq r+1$, with the property that the errors introduced by the failure only flip the parity vertices in that round. In round $j+1$, the error is detected by X stabilizers belonging to both logical patches and the routing region. Errors introduced by such failures can flip the parity of the $X \otimes X$ measurement outcome. As such, v_2 (defined in Algorithm 1) should also include the number of highlighted red space-time correlated edges incident to transition vertices.

Third, space-time correlated edges highlighted in black have the same effect for correcting errors as all other space-time correlated edges belonging to the logical patches (highlighted in green). The reason is that they are incident to parity vertices in rounds $j \geq r+2$ and thus failure mechanisms leading to such edges cannot flip the parity of the $X \otimes X$ measurement outcome.

Lastly, we note that a two-qubit gate failure arising in round $r+1$ and that results in a red highlighted space-time correlated edge will only highlight a single vertex (belonging to the logical patch in round $r+2$) throughout the entire syndrome-measurement history (assuming no other failures occur). This is due to the random outcomes of X stabilizers in round $r+1$ (so that vertices for such stabilizers cannot be highlighted in round $r+1$). Since there is an asymmetry between the number of red space-time correlated edges incident to the left data-qubit patch and those incident to the right data-qubit patch, an asymmetry in the logical failure-rate polynomials $\mathbb{P}_{(1,0,0,0)}$ and $\mathbb{P}_{(0,0,1,0)}$ (defined in Sec. III) will also arise.

APPENDIX C: RESOURCE-COST ANALYSIS OF THE HUBBARD MODEL

Following Ref. [75], the total number of logical qubits used for simulating a Hubbard model of lattice size L is $N_Q = 2L^2 + L^2/2 + 2$. If T gates are performed by catalysis, then an extra logical qubit is needed. We also add another logical qubit for the logical $|0\rangle$ required in the WTC-RFC protocol described in Sec. VI B. Using the core-cache model shown in Fig. 15, let N_1 be the number of logical qubits in the core and let N_2 be the number of logical qubits in the cache. Adding the cost of the green padding shown in Fig. 14(e), we now compute the total routing overhead cost in the core with h unit cells stacked in the vertical direction and w unit cells stacked in the horizontal direction [$h = w = 2$ in Fig. 15(d)]. We begin by defining the following functions that count the number of tiles in the green padded region of Fig. 14(e), where we separate the region into four sections:

$$\begin{aligned} s_1(d_x, h) &= h(3d_x + 1), \\ s_2(d_x, d_z, w) &= d_x + 2 + w(2d_z + d_x + 1), \\ s_3(d_x, h) &= h(3d_x + 1)(d_x + 1), \\ s_4(d_x, d_z, w) &= (d_x + 1)\Big[w(2d_z + d_x + 1) + d_x + 2\Big]. \end{aligned}$$

The total number of tiles in the green padded region is then given by

$$\begin{aligned} S_{\text{TRGP}}(d_x, d_z, h, w) = s_1(d_x, h) + s_2(d_x, d_z, w) \\ + s_3(d_x, h) + s_4(d_x, d_z, w). \end{aligned} \tag{C1}$$

The total routing overhead in the core is then

$$O^{(\text{core})}_{(d_x,d_z,h,w)} = \frac{wh(2d_z + d_x + 1)(3d_x + 1) + S_{\text{TRGP}}}{4whd_z d_x}. \tag{C2}$$

In the cache, the routing cost adds an additional $1 + (N_2 - 1)/(N_2 d_x) \sim 1 + 1/d_x$ multiplier when using surface-code patches of distance d_x and d_z. Hence, the total routing costs including both the core and the cache are given by

$$O^{(\text{total})}_{(d_x,d_z,h,w)} = \frac{\tilde{O}^{(\text{core})}_{(d_x,d_z,h,w)} + d_z[N_2(d_x+1) - 1]}{d_x d_z(4wh + N_2)}, \tag{C3}$$

where we define $\tilde{O}^{(\text{core})}_{(d_x,d_z,h,w)} = 4whd_x d_z O^{(\text{core})}_{(d_x,d_z,h,w)}$.

For the Hubbard model, the total number of logical qubits N_{TLQ}, which excludes those used in the magic state factory, is

$$N_{\text{TLQ}} = 4wh + N_2, \tag{C4}$$

with

$$N_2 = 2L^2 + \frac{L^2}{2} + 3 - 4wh, \tag{C5}$$

since there are $4wh$ logical qubits in the core. The total number of physical qubits used in the algorithm is then

$$N_{\text{phys}} = 2d_x d_z N_{\text{TLQ}} O^{(\text{total})}_{(d_x,d_z,h,w)}, \tag{C6}$$

where we use the fact that a surface-code patch can be realized using a rectangular region of size $2d_x d_z$.

We now compute the algorithm run time. Let μ be the total number of injected magic states in the core and Pauli measurements in the algorithm. Recall that in Sec. II A, μ is shown to be given by $\mu = 4N_{\text{TOF}} + N_T$. The time T_b required to inject magic states via lattice surgery is thus $T_b = \mu(d_m + 1)T_{\text{surf}}$, where T_{surf} is the time required to measure the stabilizers and reset ancillas during one surface-code syndrome-measurement cycle. Using Eqs. (4) and (6), the parameters d_x, d_z and d_m are chosen such that

$$0.01634\mu d_x \ell (21.93p)^{(d_m+1)/2} < \delta/3, \tag{C7}$$

$$0.03148\mu N_{\text{TLQ}} d_m d_x (28.91p)^{(d_z+1)/2} < \delta/3, \tag{C8}$$

$$0.0148\mu d_m \frac{\text{FA}}{d_x} (0.762p)^{(d_x+1)/2} < \delta/3. \tag{C9}$$

In Eq. (C7), we pessimistically take $d_x \ell = [d_x + 2 + h(3d_x + 1)][d_x + 2 + w(2d_z + d_x + 1)] - 4whd_x d_z$, which

TABLE I. For a Hubbard-model simulation with lattice size L and unit cells of height h and width w in the core, we provide the minimum values of d_x, d_z, and d_m such that Eqs. (C7)–(C9) are satisfied with $\delta \sim 1\%$. For the given parameters, we also provide the total number of physical qubits using Eq. (C6) and the number of logical qubits in the core and in the cache. The last column includes the multiplicative factor [defined in Eq. (C3)] that is added to the physical qubit overhead, which takes routing costs into account. The values for N_{TOF} and N_T used in computing μ are obtained from Ref. [75]. All resource costs exclude contributions from the magic state factory.

Size L	h	w	d_x	d_z	d_m	Number of physical qubits	Number of logical qubits (core)	Number of logical qubits (cache)	$O^{(\text{total})}_{(d_x,d_z,h,w)}$
						$u/\tau = 8$			
8	2	6	7	13	12	46 472	48	115	1.57
8	6	6	7	13	12	63 992	144	19	2.16
32	6	8	7	15	12	657 276	192	2371	1.23
32	14	18	7	15	12	812 532	1008	1555	1.51

is the full area of the routing space in the core. By doing so, we consider a worst-case scenario where the full routing space is used to perform lattice surgery for each injected magic state. In Eq. (C8), we use $\mathbb{P}_{Z_L} = \mathbb{P}_{(1,1,0,0)}$, since the difference with $\mathbb{P}_{(0,1,1,0)}$ is negligible. We also ignore higher-order contributions arising from lattice surgery for the reasons explained in Sec. III. Lastly, in Eq. (C9) we pessimistically set $\text{FA} = [d_x + 2 + h(3d_x + 1)][d_x + 2 + w(2d_z + d_x + 1)]$ to be the full area in the core. Such an assignment is done to take into account the possibility that the full routing space can used when performing lattice surgery after injecting a magic state. Such a scenario would lead to a large d_z distance, which would also include contributions from the logical qubits.

In Table I, we provide overhead costs associated with performing a Hubbard-model simulation of lattice size L. Given the chosen values of h and w for a unit cells in the core, we first compute the minimum required values of d_x, d_z, and d_m by solving Eqs. (C7)–(C9) with $\delta \sim 1\%$. We then compute the required number of physical qubits using Eq. (C6) and give the number of logical qubits in the core and in the cache. The last column includes the multiplicative factor resulting from the routing overhead costs. As can be seen, having more logical qubits in the core relative to those in the cache can substantially increase the routing overhead costs.

Apart from the chosen value of d_m, which satisfies Eq. (C7), the algorithm run time will depend on several factors. The first factor is the ratio of logical qubits used in the cache and in the core. Such a ratio will affect how many times one needs to read from and write to the cache during the algorithm run time. The second factor involves whether multiqubit Pauli operators with Y terms are measured using our twist-free approach or with twists. Lastly, the third factor includes run-time savings that can be achieved using our temporal-encoding scheme for fast lattice surgery. As such, a more careful analysis of the algorithm run times is left for future work. We also leave the inclusion of resource costs associated with the magic state factories to future work. However, from the results of Refs. [33,76], we expect contributions from the magic state factories to only have a mild effect on the total resource overhead costs shown in Table I.

[1] J. Preskill, Reliable quantum computers, Proc. R. Soc. London Ser. A: Math., Phys. Eng. Sci. **454**, 385 (1998).

[2] B. M. Terhal, Quantum error correction for quantum memories, Rev. Mod. Phys. **87**, 307 (2015).

[3] E. T. Campbell, B. M. Terhal, and C. Vuillot, Roads towards fault-tolerant universal quantum computation, Nature **549**, 172 (2017).

[4] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, Ann. Phys. (N. Y) **303**, 2 (2003).

[5] 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).

[6] P. W. Shor, in *Proceedings of the 37th Annual Symposium on Foundations of Computer Science* (IEEE, 1996), p. 56.

[7] R. Raussendorf, J. Harrington, and K. Goyal, Topological fault-tolerance in cluster state quantum computation, New J. Phys. **9**, 199 (2007).

[8] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, Surface code quantum computing by lattice surgery, New J. Phys. **14**, 123011 (2012).

[9] A. G. Fowler and C. Gidney, Low overhead quantum computation using lattice surgery (2018), arXiv preprint ArXiv:1808.06709.

[10] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, Poking Holes and Cutting Corners to Achieve Clifford Gates with the Surface Code, Phys. Rev. X **7**, 021029 (2017).

[11] D. Litinski and F. v. Oppen, Lattice surgery with a twist: Simplifying Clifford gates of surface codes, Quantum **2**, 62 (2018).

[12] D. Litinski, A game of surface codes: Large-scale quantum computing with lattice surgery, Quantum **3**, 128 (2019).

[13] D. Gottesman and I. L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, Nature **402**, 390 (1999).

[14] X. Zhou, D. W. Leung, and I. L. Chuang, Methodology for quantum logic gate construction, Phys. Rev. A **62**, 052316 (2000).

[15] S. Bravyi and A. Kitaev, Universal quantum computation with ideal Clifford gates and noisy ancillas, Phys. Rev. A **71**, 022316 (2005).

[16] S. Bravyi and J. Haah, Magic-state distillation with low overhead, Phys. Rev. A **86**, 052329 (2012).

[17] A. M. Meier, B. Eastin, and E. Knill, Magic-state distillation with the four-qubit code, Quant. Inf. Comp. **13**, 195 (2013).

[18] C. Jones, Multilevel distillation of magic states for quantum computing, Phys. Rev. A **87**, 042305 (2013).

[19] E. T. Campbell and M. Howard, Magic state parity-checker with pre-distilled components, Quantum **2**, 56 (2018).

[20] C. Chamberland and A. W. Cross, Fault-tolerant magic state preparation with flag qubits, Quantum **3**, 143 (2019).

[21] C. Chamberland and K. Noh, Very low overhead fault-tolerant magic state preparation using redundant ancilla encoding and flag qubits, npj Quantum Inf. **6**, 91 (2020).

[22] H. Bombin and M. A. Martin-Delgado, Topological Quantum Distillation, Phys. Rev. Lett. **97**, 180501 (2006).

[23] T. Jochym-O'Connor and R. Laflamme, Using Concatenated Quantum Codes for Universal Fault-Tolerant Quantum Gates, Phys. Rev. Lett. **112**, 010505 (2014).

[24] H. Bombín, Dimensional jump in quantum error correction, New J. Phys. **18**, 043038 (2016).

[25] T. J. Yoder, R. Takagi, and I. L. Chuang, Universal Fault-Tolerant Gates on Concatenated Stabilizer Codes, Phys. Rev. X **6**, 031039 (2016).

[26] C. Chamberland, T. Jochym-O'Connor, and R. Laflamme, Thresholds for Universal Concatenated Quantum Codes, Phys. Rev. Lett. **117**, 010501 (2016).

[27] C. Chamberland, T. Jochym-O'Connor, and R. Laflamme, Overhead analysis of universal concatenated quantum codes, Phys. Rev. A **95**, 022313 (2017).

[28] M. E. Beverland, A. Kubica, and K. M. Svore, Cost of universality: A comparative study of the overhead of state distillation and code switching with color codes, PRX Quantum **2**, 020341 (2021).

[29] S. Bravyi, G. Smith, and J. A. Smolin, Trading Classical and Quantum Computational Resources, Phys. Rev. X **6**, 021043 (2016).

[30] C. Jones, Low-overhead constructions for the fault-tolerant Toffoli gate, Phys. Rev. A **87**, 022328 (2013).

[31] B. Eastin, Distilling one-qubit magic states into Toffoli states, Phys. Rev. A **87**, 032321 (2013).

[32] C. Gidney and A. G. Fowler, Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation, Quantum **3**, 135 (2019).

[33] C. Chamberland, K. Noh, P. Arrangoiz-Arriola, E. T. Campbell, C. T. Hann, J. Iverson, H. Putterman, T. C. Bohdanowicz, S. T. Flammia, A. Keller, G. Refael, J. Preskill, L. Jiang, A. H. Safavi-Naeini, O. Painter, and F. G. S. L. Brandao, Building a fault-tolerant quantum computer using concatenated cat codes (2020), arXiv preprint ArXiv:2012.04108.

[34] E. T. Campbell and M. Howard, Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost, Phys. Rev. A **95**, 022316 (2017).

[35] E. T. Campbell and M. Howard, Unifying Gate Synthesis and Magic State Distillation, Phys. Rev. Lett. **118**, 060501 (2017).

[36] J. Haah and M. B. Hastings, Codes and protocols for distilling T, controlled-S, and Toffoli gates, Quantum **2**, 71 (2018).

[37] C. Gidney, Halving the cost of quantum addition, Quantum **2**, 74 (2018).

[38] J. O'Gorman and E. T. Campbell, Quantum computation with realistic magic-state factories, Phys. Rev. A **95**, 032338 (2017).

[39] E. Campbell, A. Khurana, and A. Montanaro, Applying quantum algorithms to constraint satisfaction problems, Quantum **3**, 167 (2019).

[40] D. W. Berry, C. Gidney, M. Motta, J. R. McClean, and R. Babbush, Qubitization of arbitrary basis quantum chemistry leveraging sparsity and low rank factorization, Quantum **3**, 208 (2019).

[41] 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).

[42] J. Lee, D. W. Berry, C. Gidney, W. J. Huggins, J. R. McClean, N. Wiebe, and R. Babbush, Even more efficient quantum computations of chemistry through tensor hypercontraction, PRX Quantum **2**, 030305 (2021).

[43] Y. Tomita and K. M. Svore, Low-distance surface codes under realistic quantum noise, Phys. Rev. A **90**, 062320 (2014).

[44] C. Vuillot, L. Lao, B. Criger, C. G. Almudéver, K. Bertels, and B. M. Terhal, Code deformation and lattice surgery are gauge fixing, New J. Phys. **21**, 033028 (2019).

[45] R. Raussendorf and J. Harrington, Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions, Phys. Rev. Lett. **98**, 190504 (2007).

[46] A. G. Fowler, A. M. Stephens, and P. Groszkowski, High-threshold universal quantum computation on the surface code, Phys. Rev. A **80**, 052312 (2009).

[47] J. Edmonds, Paths, trees, and flowers, Can. J. Math. **17**, 449 (1965).

[48] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits, Phys. Rev. X **10**, 011022 (2020).

[49] C. Chamberland, A. Kubica, T. J. Yoder, and G. Zhu, Triangular color codes on trivalent graphs with flag qubits, New J. Phys. **22**, 023019 (2020).

[50] Note, however, that if the numbers of syndrome-measurement rounds both before and after the merge are identical, we expect $(1,0,0,0)$ and $(1,1,0,0)$ events to have similar failure probabilities.

[51] H. Bombin, Topological Order with a Twist: Ising Anyons from an Abelian Model, Phys. Rev. Lett. **105**, 030403 (2010).

[52] T. J. Yoder and I. H. Kim, The surface code with a twist, Quantum **1**, 2 (2017).

[53] An analysis for the impact of twists on the run time and performance of lattice surgery will appear in future work.

[54] R. Chao and B. W. Reichardt, Quantum Error Correction with Only Two Extra Qubits, Phys. Rev. Lett. **121**, 050502 (2018).

[55] R. Chao and B. W. Reichardt, Fault-tolerant quantum computation with few qubits, npj Quantum Inf. **4**, 2056 (2018).

[56] C. Chamberland and M. E. Beverland, Flag fault-tolerant error correction with arbitrary distance codes, Quantum **2**, 53 (2018).

[57] Y. Shi, C. Chamberland, and A. Cross, Fault-tolerant preparation of approximate GKP states, New J. Phys. **21**, 093007 (2019).

[58] T. Tansuwannont, C. Chamberland, and D. Leung, Flag fault-tolerant error correction, measurement, and quantum computation for cyclic Calderbank-Shor-Steane codes, Phys. Rev. A **101**, 012342 (2020).

[59] R. Chao and B. W. Reichardt, Flag fault-tolerant error correction for any stabilizer code, PRX Quantum **1**, 010302 (2020).

[60] B. W. Reichardt, Fault-tolerant quantum error correction for Steane's seven-qubit color code with few or no extra qubits, Quantum Sci. Technol. **6**, 015007 (2020).

[61] T. Tansuwannont and D. Leung, Achieving fault tolerance on capped color codes with few ancillas (2021), arXiv e-prints, eid ArXiv:2106.02649.

[62] A. G. Fowler, Time-optimal quantum computation (2012), arXiv preprint ArXiv:1509.03239.

[63] I. H. Kim, E. Lee, Y.-H. Liu, S. Pallister, W. Pol, and S. Roberts, Fault-tolerant resource estimate for quantum chemical simulations: Case study on Li-ion battery electrolyte molecules (2021), arXiv preprint ArXiv:2104.10653.

[64] S. Puri, L. St-Jean, J. A. Gross, A. Grimm, N. E. Frattini, P. S. Iyer, A. Krishna, S. Touzard, L. Jiang, and A. Blais, *et al.*, Bias-preserving gates with stabilized cat qubits, Sci. Adv. **6**, eaay5901 (2020).

[65] E. T. Campbell, A theory of single-shot error correction for adversarial noise, Quantum Sci. Technol. **4**, 025006 (2019).

[66] A. O. Quintavalle, M. Vasmer, J. Roffe, and E. T. Campbell, Single-shot error correction of three-dimensional homological product codes, PRX Quantum **2**, 020340 (2021).

[67] D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, Fault-Tolerant Thresholds for the Surface Code in Excess of 5% under Biased Noise, Phys. Rev. Lett. **124**, 130501 (2020).

[68] A. Ashikhmin, C.-Y. Lai, and T. A. Brun, Quantum data-syndrome codes, IEEE J. Sel. Areas Commun. **38**, 449 (2020).

[69] D. K. Tuckett, A. S. Darmawan, C. T. Chubb, S. Bravyi, S. D. Bartlett, and S. T. Flammia, Tailoring Surface Codes for Highly Biased Noise, Phys. Rev. X **9**, 041031 (2019).

[70] J. P. Bonilla Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, The XZZX surface code, Nat. Commun. **12**, 2172 (2021).

[71] A. S. Darmawan, B. J. Brown, A. L. Grimsmo, D. K. Tuckett, and S. Puri, Practical quantum error correction with the XZZX code and Kerr-cat qubits (2021), arXiv e-prints ArXiv:2104.09539.

[72] O. Higgott and N. P. Breuckmann, Subsystem Codes with High Thresholds by Gauge Fixing and Reduced Qubit Overhead, Phys. Rev. X **11**, 031039 (2021).

[73] N. P. Breuckmann, C. Vuillot, E. Campbell, A. Krishna, and B. M. Terhal, Hyperbolic and semi-hyperbolic surface codes for quantum storage, Quantum Sci. Technol. **2**, 035007 (2017).

[74] J. Conrad, C. Chamberland, N. P. Breuckmann, and B. M. Terhal, The small stellated dodecahedron code and friends, Phil. Trans. R. Soc. A **376**, 20170323 (2018).

[75] E. T. Campbell, Early fault-tolerant simulations of the Hubbard model (2020), ArXiv:2012.09238.

[76] D. Litinski, Magic state distillation: Not as costly as you think, Quantum **3**, 205 (2019).