Scalable Surface-Code Decoders with Parallelization in Time

Xinyu Tan,[†] Fang Zhang,[†] Rui Chao, Yaoyun Shi, and Jianxin Chen[*]
Quantum Laboratory, DAMO Academy, Bellevue, Washington 98004, USA

(Received 29 September 2022; accepted 8 November 2023; published 15 December 2023)

Fast classical processing is essential for most quantum fault-tolerance architectures. We introduce a sliding-window decoding scheme that provides fast classical processing for the surface code through parallelism. Our scheme divides the syndromes in space-time into overlapping windows along the time direction, which can be decoded in parallel with any inner decoder. With this parallelism, our scheme can solve the backlog problem as the code scales up, even if the inner decoder is slow. When using minimum-weight perfect matching and union find as the inner decoders, we observe circuit-level thresholds of 0.68% and 0.55%, respectively, which are almost identical to those for the batch decoding.

DOI: 10.1103/PRXQuantum.4.040344

I. INTRODUCTION

Fault-tolerance theory allows scalable and universal quantum computation provided that the physical error rates are below a threshold. Aharonov and Ben-Or, in their seminal work [1], give a fault-tolerance scheme with no classical operations. However, the resulting threshold is far from feasible. Multiple fault-tolerance architectures have since been proposed [2–4] but their estimations for the threshold and resource overhead mostly assume instantaneous classical computations.

One prominent architecture [5,6] uses the surface code [7] and achieves universality via magic state distillation [8]. In particular, the implementation of a non-Clifford gate using a magic state typically involves a classically controlled Clifford correction. Therefore, the error correction between consecutive non-Clifford gates should be fast enough to keep up with the rapidly decohering quantum hardware, so that the error syndromes do not backlog [9]. This requires an adequately high decoding throughput—the amount of error syndromes that can be processed by a decoder in unit time.

Many decoding schemes for the surface code have high thresholds [10,11], yet they cannot be implemented with a high enough throughput and thus do not address the backlog problem. On the other hand, local decoding schemes [12–19] are fast and scalable to a certain degree but their speed comes at the expense of accuracy. The accuracy of local decoders can be improved by appending a global decoder [20–25] while still pursuing relatively high decoding throughputs. Other schemes based on specialized hardware [26–29] have also been proposed. However, to the best of our knowledge, none of the approaches mentioned above have demonstrated adequate accuracy, throughput, and scalability simultaneously.

In this work, we introduce the *sandwich decoder* for the surface code, which solves the backlog problem using parallelism. Our work is inspired by the idea of "overlapping recovery" in Ref. [5] (later rediscovered in Ref. [30]), which we reformulate as the *forward decoder*. Both the sandwich and forward decoders are *sliding-window decoders*, i.e., they divide the error syndromes in space-time into overlapping windows in the time direction. As opposed to the sliding-window decoders, we define *batch decoders* as the algorithms that process the error syndromes all at once, i.e., there exists only one window.

One significant limitation of the forward decoder is that it processes windows sequentially, resulting in a limited throughput. However, our sandwich decoder removes the dependency between the windows so that they can be handled in parallel, e.g., using separate classical processing units [31]. Adjacent sandwich windows may diagnose differently upon the same syndromes, which would compromise the fault-tolerance property of the decoding scheme. We reconcile such inconsistency by decoding the controversial syndromes in a further subroutine.

The parallelism of the sandwich decoder is a great advantage for scalability. An inherently sequential algorithm like the forward decoder can hardly take advantage of parallel computational resources and thus will have difficulty maintaining adequate throughput when the code

[*]chenjianxin@tsinghua.org.cn
[†]These two authors contributed equally to this work.

distance increases. Meanwhile, the sandwich decoder can solve the backlog problem as the code scales up, as long as it is given enough parallel processing units. Little communication is needed between processing units, as there is no dependency between windows. Thus the throughput requirement can be easily satisfied by adding more cores or processors, which is much easier than pushing the processor clock speed. Furthermore, the number of parallel processing units needed only scales with the speed of the quantum hardware and the code distance, not with the length of the quantum computation.

We benchmark the sandwich decoder with the memory experiment for the distance-d-rotated surface code [32] under circuit-level noise. In particular, we decode each window using the minimum-weight perfect-matching (MWPM) [5] or union-find (UF) decoder [33] and observe numerical thresholds of 0.68% and 0.55%, respectively, for the logical error rate per d cycles of syndrome extraction. These values are almost identical to the corresponding thresholds for the batch decoders. It is reasonable to expect similar preservation of accuracy when using other inner decoders. In consequence, our sandwich decoder may allow us to prioritize the accuracy of the inner decoder, as throughput is assured simply with adequate computational resources.

The paper is organized as follows. In Sec. II, we review the concept of decoder graphs, with special focus on different types of boundaries in the decoder graph. In Sec. III, we build a theoretical foundation for sliding-window decoders. We note that existing designs of sliding-window decoders in the literature are in fact forward decoders under our classification scheme and thus cannot efficiently utilize the parallelism between windows. In Sec. IV, we introduce the sandwich decoder, which, in contrast to the forward decoder, enables scalable parallelization between windows. In Sec. V, we analyze the performance of the sandwich decoder in terms of both the asymptotic behavior of the decoding throughput and logical error rates and thresholds observed in experiments. We talk about some possible directions to generalize our results in Sec. VI, before giving some concluding remarks in Sec. VII.

II. DECODER GRAPHS WITH BOUNDARIES

A. The memory experiment

We focus on the memory experiment that preserves the logical state $|\overline{0}\rangle$ for the $[\![d^2, 1, d]\!]$-rotated surface code; the argument for other variants of the surface code or logical basis states proceeds analogously. Specifically, we first initialize each data qubit as the state $|0\rangle$. Then, we repeatedly apply a syndrome-extraction circuit for n cycles and obtain syndromes $\sigma_i^X, \sigma_i^Z \in \{0,1\}^{(d^2-1)/2}$ of the X- and Z-type check operators, respectively, for $i = 1, \ldots, n$. Finally, we measure all the data qubits in the Z basis and obtain outcomes $\mu \in \{0,1\}^{d^2}$.

For the surface code, each cycle of *detectors* is the exclusive OR (XOR) of two consecutive cycles of syndromes. More precisely, let $\sigma_{n+1}^Z(\mu) \in \{0,1\}^{(d^2-1)/2}$ be the syndromes of the Z-type check operators evaluated from the data-qubit measurement outcomes μ. Define

$$\begin{aligned} \delta_1^Z &:= \sigma_1^Z, \\ \delta_i^P &:= \sigma_i^P \oplus \sigma_{i-1}^P, \quad P \in \{X, Z\},\ i = 2, 3, \ldots, n, \text{ and} \\ \delta_{n+1}^Z &:= \sigma_{n+1}^Z(\mu) \oplus \sigma_n^Z. \end{aligned} \tag{1}$$

We further assume that the syndrome-extraction circuit is fault tolerant [32] and the whole circuit of the memory experiment is afflicted with stochastic Pauli errors. Specifically, each gate, qubit idling, and initialization (respectively, measurement) is modeled as the ideal operation followed (respectively, preceded) by a random Pauli, referred to as a *fault*, supported on the involved qubit(s).

Under our assumptions about the circuit and error model, detectors are 0 in the absence of faults; thus, any *defects*—detectors with value 1—indicate the presence of faults. Furthermore, the occurrence of each fault flips at most two detectors of each type (X or Z) [32]. We define a detector to be *open* if there is a fault that flips that detector but no other detector of the same type; otherwise, it is *closed*.

Example 1.—In Fig. 1(a), an X fault on data qubit A flips detectors α and β; whereas an X fault on data qubit B only flips detector γ. γ is an open detector and α and β are closed detectors.

Figure 1(b) illustrates a *Z-type decoder graph* constructed as follows. First, add one vertex for each Z-type detector. Then, add an edge between two vertices (detectors) if there is a fault that flips them both. Finally, for each open detector, add an *imaginary detector* and an edge connecting them. We also assign each imaginary detector a binary value, such that each fault flips either zero or two Z-type detectors. Each edge in the decoder graph thus represents an equivalence class of faults that flip the same two detectors.

The goal of a decoding procedure is to *annihilate* all defects by finding a proper set of *corrections*—edges that give rise to the exact same defects. Formally, a defect is annihilated if it is incident to an odd number of edges in the set.

B. Open and closed boundary conditions

The notions "open" and "closed" can be extended to the boundaries of the three-dimensional (3D) decoder graph. There are four *space boundaries* and two *time boundaries*. Each space boundary consists of all the real detectors adjacent to the top-, bottom-, left-, or right-most [in the

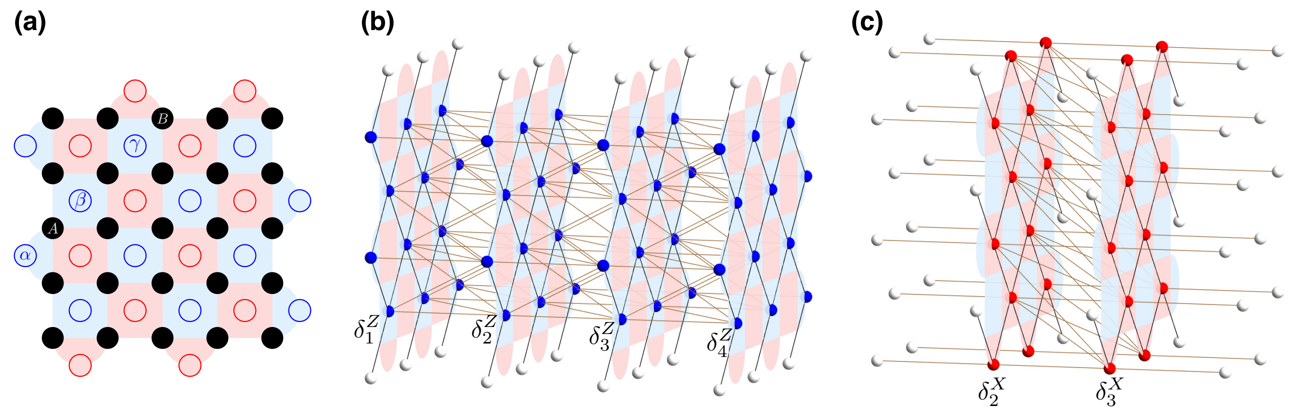

FIG. 1. (a) A distance-5-rotated surface code. The data qubits (black) reside on the plaquette corners. The check operators of types Z (blue) and X (red) are measured with ancilla qubits (empty circles) on the plaquette centers. (b),(c) Decoder graphs of types (b) Z and (c) X for a memory experiment with three syndrome-extraction cycles that preserves $|\bar{0}\rangle$ of the code shown in (a). The blue and red vertices denote Z- and X-type real detectors, respectively; the white vertices denote imaginary detectors. Each edge represents the set of faults that flip the incident detectors. Each decoder graph has two open space boundaries, whereas the X-type decoder graph also has two open time boundaries [34].

directions of Fig. 1(a)] [35] data qubits of each layer, respectively. The first and last layers of real detectors are the time boundaries, representing the detectors at the time of data-qubit initialization and final data-qubit measurements.

A boundary in a decoder graph is called open if every detector on this boundary is open. A boundary is closed if it is not open.

1. Space boundaries

Let us consider the Z-type decoder graph with the layout specified in Fig. 1(b). An X fault on any of the data qubits on the top and bottom boundaries of the lattice in Fig. 1(a) flips only one real Z-type detector (i.e., a bit in δ^Z), which is open by definition. Hence, the top and bottom space boundaries of the 3D decoder graph in Fig. 1(b) can be referred to as open and the left and right space boundaries are closed.

Remark 1.—One way to intuitively justify the words “open” and “closed” is by looking at the forms of undetectable errors. For codes without space boundaries (such as the toric code), an undetectable error always looks like a cycle or a combination of cycles, either topologically trivial (in which case it will never cause a logical error) or not (in which case it may be a logical operator). For codes with space boundaries, an undetectable error can also be a path with both ends at the open boundaries, as if the path goes into and out of the code patch through those boundaries.

2. Time boundaries

Similarly, both the past (left) and future (right) time boundaries in Fig. 1(b) are closed, since no fault flips only one detector in δ_1^Z or δ_{n+1}^Z (unless on a space boundary).

Example 2.—Suppose that there is only a Z-stabilizer measurement error during the last cycle of syndrome extraction, i.e., $\sigma_i^Z = 0$ for $i \in \{1,\ldots,n+1\} \setminus \{n\}$ and $\sigma_n^Z = 0\cdots010\cdots0$. It follows from Eq. (1) that there are only two defects in δ_n^Z and δ_{n+1}^Z, respectively. The edge connecting these two defects indicates the Z-stabilizer measurement error.

Example 3.—Suppose that there is only a data-qubit measurement error at the end of the memory experiment. We have $\sigma_i^Z = 0$ for $i \in \{1,\ldots,n\}$. As we calculate the final set of Z syndromes, σ_{n+1}^Z, based on the data-qubit measurement results μ, one flipped data qubit affects all the check operators that it involves. Therefore, σ_{n+1}^Z has one or two nontrivial syndromes and δ_{n+1}^Z has one or two defects. With the imaginary detectors on the open space boundaries, any single data-qubit measurement fault flips exactly two detectors.

However, for the X-type decoder graph [see Fig. 1(c)], the outcome of the first cycle of X-stabilizer extraction is a random binary string even if there are no errors. Therefore, the first round of X-type real detectors is formed from the parity of the first two rounds of syndrome-measurement outcomes, which are trivial in the absence of faults. In particular, the first round of X-type detectors forms an open time boundary. That is, each of the X-type detectors in the first round is connected to an imaginary detector, in order to cope with error mechanisms that only flip a detector in this first round. Similarly, the ending time boundary also needs to be open, since in the end, after we measure the data qubits in the Z basis, the outcome μ does not provide any information about the X syndromes.

3. Importance

We emphasize that the open and closed boundary conditions are not just some mathematical tricks that marginally improve the performance of the decoder; on the contrary, to get meaningful results from the quantum memory experiment, one must correctly close or open the boundaries according to the context (e.g., the code or the specific type of fault).

As we focus on the memory experiment of preserving logical $|\bar{0}\rangle$, our goal is to prevent the logical Z operator from being flipped (an odd number of times). But if one of the time boundaries in the Z-type decoder is open, there will be low-weight (i.e., short) undetectable X errors with both end points on that time boundary. Such an undetectable X error can easily flip the logical Z operator, violating the principle that only at least $\lceil d/2 \rceil$ physical faults can cause a failure. On the other hand, when both time boundaries are closed, everything makes sense: the only open boundaries are the top and bottom space boundaries and low-weight X fault chains starting and ending at one of those boundaries can only flip the logical Z operator an even number of times. To flip the logical Z operator, fault chains must cross from one open space boundary to the other open space boundary but then it is a logical operator with weight $\geq d$ and all is well.

We provide another way to understand the open-time-boundary condition in Sec. III A and show with numerical evidence in Appendix B that misusing closed- and open-time-boundary conditions greatly increases the logical error rate.

III. DESIGNING SLIDING-WINDOW DECODERS

A *window* consists of a number of consecutive cycles of detectors in the decoder graph [see Fig. 2(a)]. Both the forward and sandwich decoders work on one window at a time. Each window is processed by an *inner decoder*, a subroutine that generates corrections for that window only. The corrections assembled from all windows should collectively annihilate every observed defect in the whole decoder graph.

Hereafter, we associate each window with a subgraph of Fig. 1(b) consisting of the real detectors enclosed in the window and certain imaginary detectors. Concretely, the first window contains detectors from δ_1^Z to δ_w^Z, the second window from δ_{1+s}^Z to δ_{w+s}^Z, and so forth. That is, windows, each with length w, proceed rightward with step size $s < w$. The final window contains detectors up to δ_{n+1}^Z.

A. Artificial time boundaries

The idea of allocating detectors in windows naturally creates two *artificial time boundaries* for each window (except for the first and final ones, each of which has one real time boundary and one artificial time boundary). Note that these artificial time boundaries do not represent any real initialization or termination of the memory experiment. Therefore, they should naturally be open, indicating that there may still be detectors at the other side of each artificial time boundary unknown to the current window.

From the perspective of the current window, an isolated defect close to such a time boundary is more likely to be caused by some faults from the future or the past. However, if the inner decoder regards this boundary as closed, it will be forced to generate corrections of a higher weight within the current window, which is more likely to cause a logical error in the final result.

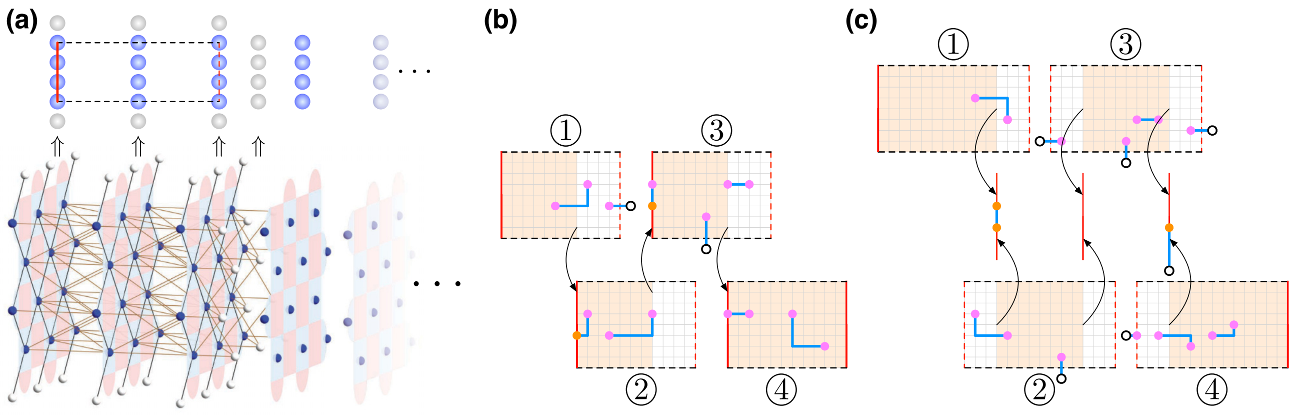

FIG. 2. (a) The decoder graph of a first window of length $w = 3$. As in Fig. 1(b), the blue and white vertices denote Z-type real and imaginary detectors, respectively. Each edge represents the set of faults that flip the incident detectors. The past time boundary (red line) is closed; the top and bottom space boundaries (black dashed line) and the future time boundary (red dashed line) are open. Imaginary detectors near the future time boundary are equivalent to vertices in δ_4^Z. (b),(c) The (b) forward decoder and (c) sandwich decoder. The observed defects (pink dots) are annihilated by the corrections (blue lines). Corrections in the cores (brown region) are retained, whereas those in the buffers (transparent region) are discarded. Retained corrections can create updated defects (orange dots) on the seams between adjacent cores. Imaginary detectors reside near the open boundaries (dashed line) and only those incident to corrections are shown (empty circle). The forward decoder must decode the windows sequentially by the order labeled, whereas the sandwich decoder can, in principle, decode the windows in parallel.

In some cases, an artificial time boundary can also become close with some modifications, such as the artificial past time boundaries in the forward decoder. This is also one of the key differences between the forward and sandwich decoders.

B. Core regions and buffer regions

In both forward and sandwich decoders, each inner decoder takes all the detectors within a window as input and returns corrections for the entire window. However, since adjacent windows generally have an overlap, we do not need to apply all these corrections. Instead, each window only retains a part of the corrections that are relatively reliable and disregards the rest. The former region (where corrections are accepted) is called the *core region* and the latter region (the rest of the window) is called the *buffer region*. In general, the size of a core region equals the step size. The buffer regions are usually close to the open artificial time boundaries, since their corrections are less trustworthy. We will be more precise about these two regions later.

Intuitively, the buffer regions between windows let them share precious contextual information on faults with each other. Therefore, having a large buffer is beneficial when we merge the individual corrections generated by each inner decoder back to the entire decoder graph, since the accepted corrections become more reliable.

C. Correction consistency

One important principle of surface codes is that the "correct" corrections are not unique: any two sets of corrections that differ by one or more stabilizers are logically equivalent. This fact poses a challenge for sliding-window decoders: even if each decoder window individually finds a "correct" set of corrections, there may not be an obvious method to combine them into a consistent set of corrections for the entire decoder graph.

An example of this is illustrated in Fig. 2(c): the windows labeled ① and ② return inconsistent corrections along the "seam" where we want to merge. The possibility of such an inconsistency means that the combined corrections may not annihilate all defects and that even a low-weight fault may cause a logical error.

D. Forward-window decoder

The main idea of a forward decoder is to sequentially decode each window one at a time and then propagate necessary syndrome information from the current window to the next window. In this way, it solves the inconsistency problem by forcing the next window to output consistent corrections with the current window.

In Fig. 2(b), for each window (not the final one) that spans the detectors from δ^Z_{1+is} to δ^Z_{w+is}, let its past boundary be closed and its future boundary be open. Also, let the core region be the set of edges in the window that are incident to at least one vertex ranging from δ^Z_{1+is} to $\delta^Z_{(i+1)s}$ and let the buffer region be the remaining edges in the window. For the final window, both time boundaries are closed and all edges belong to the core. The core regions in two adjacent windows overlap in exactly the vertices on the past boundary of the later window.

The forward decoder processes the windows in temporal order. Within each window starting from δ^Z_{1+is}, the inner decoder first finds a set of corrections that can annihilate the existing defects but only retains the corrections in the core region. If the current window is the final one, all observed defects will have been annihilated and the decoder will terminate. Otherwise, only the defects from δ^Z_{1+is} to $\delta^Z_{(i+1)s}$ will be annihilated and the detectors on $\delta_{1+(i+1)s}$ will be updated and deferred to the next window.

In this way, since all the defects prior to the current window have been annihilated by previously accepted corrections, it has a closed past boundary and an open future boundary. As a result, a forward window needs no buffer preceding the core. The closed boundary means that the inner decoder will not change any corrections already accepted in the past and thus consistency between windows is ensured.

The corrections found in the core become more reliable with larger buffer size, as the future faults outside the window are less likely to affect the core.

As far as we know, the idea of the forward decoder first appears in Ref. [5] with $w = 2s$ and is rediscovered in Ref. [30] with $s = 1$ but without specifying the future-boundary conditions of windows.

E. Limitation with parallelization

A remarkable disadvantage of the forward approach is the strict dependency among all windows: we cannot start decoding the next window until the current window is finished. Therefore, once the decoding time of each window fails to keep up with s cycles of newly extracted syndromes, where s is the step size, the latency accumulates. This is one of the reasons why the authors of Ref. [30] employs look-up tables (LUTs) as their inner decoders. Since the LUTs are generated off-line (or in advance) using MWPM, they are fast and accurate while decoding each window; however, LUTs require a huge amount of memory for storage, restricting code distances to up to 5 and window size to a maximum of 3.

For sliding-window decoders, the throughput can be greatly increased if we can decode all windows in parallel. However, this is impossible for a forward decoder, since this data dependency causes the critical path [36] to run through all windows.

IV. SANDWICH-WINDOW DECODER

Next, we introduce the sandwich decoder, the windows of which can be processed effectively in parallel. It has a much shorter critical path that does not increase with the total number of windows [37]. Figures 2(b) and 2(c) visualize the difference between the forward approach and the sandwich approach.

The sandwich decoder has two subroutines and each subroutine deals with a specific type of sandwich window. If we do not specify the type of a window, it should be clear from the context.

The reason for naming this approach as the *sandwich* decoder is twofold, as we shall explain later. We discuss in Sec. VI other generalizations of the sandwich decoder, especially the potential to adapt it to the stability experiment [38] and to lattice surgery [39].

A. Type-1 windows: Buffer regions in both directions

By propagating the syndromes in the forward direction, the forward decoder can solve the correction consistency problem but with a long critical path. Since the decoder-graph formalism is symmetric with respect to the direction of time, each window should be capable of propagating syndromes in the backward direction as well.

It follows naturally that except for the first and final windows, the core region of each window can be "sandwiched" by two buffer regions and both artificial time boundaries are open. The first (respectively, final) window has only one buffer and one open time boundary in the future (respectively, past) direction. We refer to such a window as a type-1 sandwich window and $w = s + 2b$, where b is the length of the buffer region. Although adjacent type-1 sandwich windows overlap, they can be regarded as independent and thus all of them can be decoded in parallel.

There is a lot of freedom regarding the detailed design. Here, we describe one choice formally. For each window that spans the detectors from δ^Z_{1+is} to δ^Z_{w+is}, the core region contains all edges incident to at least one vertex ranging from δ^Z_{b+1+is} (δ_1 if initial window) to $\delta^Z_{s+b-1+is}$ (δ^Z_{n+1} if final window). Each inner decoder finds corrections that can annihilate all defects within the window but only retains the ones in the core.

B. Type-2 windows: Merging corrections

Since type-1 windows propagate syndrome information forward and backward, there must be another type of window that receives syndrome information from both ends, which we define as the type-2 window. Since syndrome propagation closes the corresponding time boundary of the receiving window, each type-2 sandwich window has *both* time boundaries closed and the entire window is the core region.

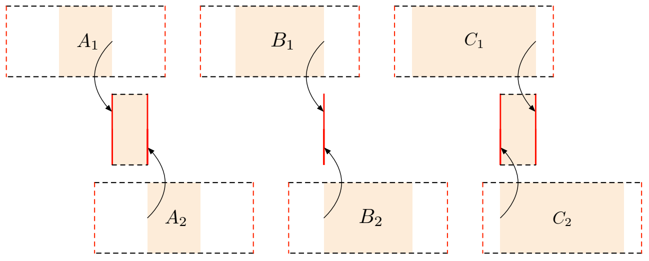

FIG. 3. Positive (A), zero (B), and negative (C) seam offsets.

Since each type-2 window is sandwiched between two type-1 windows, it is dependent on the decoding results of the two independent type-1 windows. But since all type-2 windows are independent from each other, they can be decoded in parallel.

After we apply the corrections from type-1 windows, there may exist defects that have not been annihilated yet. The role of type-2 windows is to reconcile this inconsistency by neutralizing *all* remaining defects.

To illustrate this more clearly, we call the latest layer of detectors in a core region the *right seam* and the oldest layer the *left seam*. The core regions of two adjacent type-1 windows can have three patterns (Fig. 3) and we refer to this difference using a parameter called the *seam offset*:

(a) When the seam offset is positive, the core regions do not have any overlap.
(b) When the seam offset is 0, the right seam of the former core overlaps with the left seam of the latter core. The type-2 windows thus become two-dimensional (2D) and the contained detectors have been updated twice. For simplicity of illustration and implementation, we set the seam offset to be 0 in the numerical analysis and in the visualization in Fig. 2(c).
(c) When the seam offset is negative, the core regions overlap in more than one layer.

In each case, the size of a type-2 window is $|t| + 1$, where t is the seam-offset value. A type-2 window is 3D unless the seam offset is 0.

We can always find valid corrections for each type-2 window, since its top and bottom space boundaries are open. After decoding all type-2 windows and applying every correction, all defects will be annihilated.

V. ANALYSIS

A. Throughput

Sliding-window decoders may induce a throughput overhead compared to batch decoders, due to the overlap between windows. However, with reasonable parameter choices, this overhead is only a minor constant factor compared to the substantial potential for increased throughput through parallelism.

Assuming that the time complexity of decoding a window is linear in the window volume, this overhead factor is equal to the ratio of the window size to the core region volume, i.e., w/s, where w is the window size and s is the step size. For UF decoders, the linear decoding complexity holds true in practice; for decoders with superlinear complexity, such as a naive version of the MWPM decoder, dividing the decoding graph into windows even reduces the asymptotic time complexity.

For the sandwich decoder, where $w = s + 2b$, the ratio can be made arbitrarily close to 1 by increasing the window size, at the cost of increased decoding scheme latency. In practice, setting the overhead factor to a modest value of 2 by taking $w = 4b$ is sufficient. With 1000 available CPU cores for decoding, the sandwich decoder can still achieve a 500-fold throughput increase compared to the unparallelized batch decoder.

B. Accuracy and threshold

We benchmark our sandwich decoder for odd code distances $d = 3, 5, \ldots, 17$ with step size $s = (d + 1)/2$, window size $w = 3s$, and varying number of cycles n. We employ the independent depolarizing error model with rate p for the entire circuit (for the detailed method of evaluating the thresholds, see Appendix A).

Since n typically scales with d, a natural metric for accuracy is the *logical error rate per d cycles*, $p_L(d)$, assuming the ansatz that each cycle of syndrome extraction independently flips the logical qubit with a fixed probability. We first calculate $p_L(d)$ by fitting the logical error rates obtained from Monte Carlo simulations with varying n. Then, we determine the threshold of the physical error rate p according to the trend of $p_L(d)$.

With the above procedure, we observe threshold values $p = 0.680(2)\%$ and $0.554(1)\%$ for the MWPM and UF decoders as inner decoders, respectively (see Fig. 4). For comparison, we also run the same experiments with batch decoding instead of sandwich decoding, i.e., decoding all the syndromes of a memory experiment as a single window. We observe $p_L(d)$ close to but systematically lower than those with sandwich decoding and thresholds of 0.683(2)% and 0.553(2)% for the respective inner decoders.

C. Exploring existing freedoms

There are many freedoms in our sliding-window decoding scheme and we explore some of them in Appendix B. For example, we vary the step size, window size, and seam offset; compare sandwich and forward windows; compare open and closed time boundaries; and apply sandwich-window decoders to real-world data provided in Ref. [40].

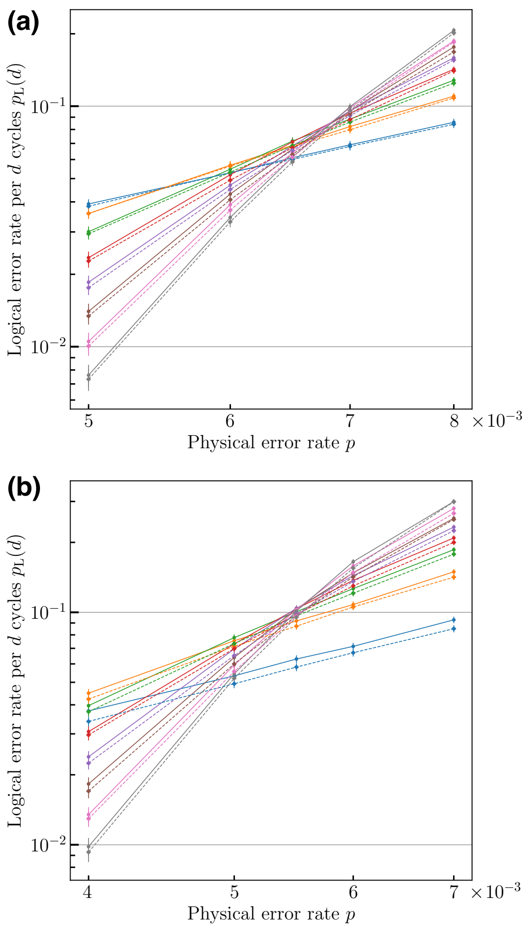

FIG. 4. A comparison between sandwich decoding (solid line) and batch decoding (dashed line) with (a) MWPM and (b) UF decoders as the inner decoder. For fixed code distance $d = 3, 5, \ldots, 17$ and physical error rate p, we first vary the number of cycles of syndrome extraction n and simulate each memory experiment for 10^5 shots. Then, we collect the estimated logical error rates per shot for varying n and calculate the logical error rate per d cycles $p_L(d)$, depicted as dots. The error bars indicate 95% statistical confidence and dots of the same d value are connected for ease of visualization.

Insights into how other freedoms may affect the performance of our scheme will be valuable. For example, regarding the UF inner decoder, using a peeling decoder based on breadth-first search instead of depth-first search may affect the effectiveness of the syndrome propagation process.

VI. GENERALIZATIONS

A. Stability experiment

One omission from this paper is the *stability experiment* as described in Ref. [38]. One of the main motivations for the stability experiment is to emulate the "space-like parts" that arise in various useful logical operations with lattice surgery, such as moving a qubit or doing

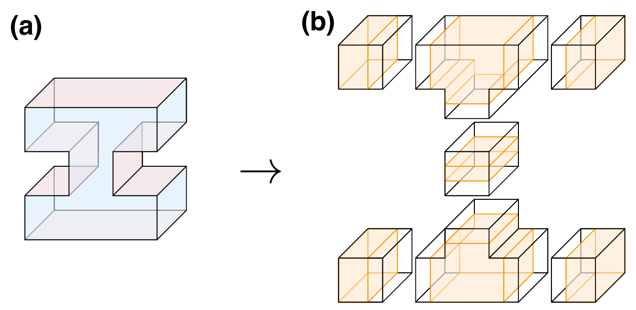

FIG. 5. An illustration of how our scheme can be potentially extended to lattice-surgery operations such as the two-qubit measurement. (a) The shape of the 3D decoder graph for an experiment containing a two-qubit measurement. (b) A way to divide this decoder graph into windows, with the core and buffer regions for each window illustrated. Each window has size $O(d) \times O(d) \times O(d)$.

a two-qubit parity measurement. Ideally, each of those "spacelike parts" should last only for $O(d)$ surface-code cycles, since adding more cycles has diminishing returns for suppressing timelike logical errors and is detrimental for suppressing spacelike logical errors (of the opposite X/Z type). On the other hand, the spatial span of a "spacelike part" may be significantly larger than d, depending on the physical distance on the surface-code lattice between the qubits involved.

Thus, it is reasonable to consider stability experiments on an elongated rectangular code patch and divide it into windows in a spatial direction instead of the temporal direction. Such a sliding-window decoder would not be fundamentally different from a sliding-window decoder for the memory experiment, only with the roles of the time and one spatial dimension switched.

B. Lattice surgery

We can then generalize our sliding-window decoders to some more useful operations in lattice surgery. For example, the two-qubit parity measurement has an overall H-shaped decoder graph, as opposed to the rectangular-box–shaped decoder graph for the memory experiment (where the box is elongated in the temporal direction) or the stability experiment (where the box is elongated in a spatial direction), but it is still straightforward to divide the graph into 3D windows each with dimensions $O(d) \times O(d) \times O(d)$, as shown in Fig. 5. Two T-shaped windows simply need to propagate seam syndromes in three directions instead of two. Other lattice-surgery operations may add more complexity to the scheme—e.g., a twist defect may require combining the X-decoder graph and the Z-decoder graph in some way—but it seems that the same principle should be able to handle everything.

C. Parallelization on hypergraphs

A particularly straightforward avenue for generalization is any stabilizer code the decoding problem for which can be formulated through a hypergraph. Concretely, given a stabilizer code and its syndrome-extraction circuit, denote by $\mathcal{V}$ the set of detectors. Then, a stochastic Pauli error model induces a hypergraph $(\mathcal{V}, \mathcal{E})$ with

$$\mathcal{E} = \{e \subseteq \mathcal{V} \,:\, \text{there is a fault that flips exactly the detectors in } e\}. \tag{2}$$

Consider the $\mathbb{F}_2$-linear map from the edge space to the vertex space $\partial : \mathbb{F}_2^{|\mathcal{E}|} \to \mathbb{F}_2^{|\mathcal{V}|}$,

$$\partial E := \sum_{e\in E} \sum_{v\in e} v, \tag{3}$$

where the vector addition corresponds to symmetric difference. For each $V \subseteq \mathcal{V}$ and $E \subseteq \mathcal{E}$, define

$$\Delta(E, V) := \{e \in E \,:\, e \text{ incident to a vertex in } V\}\,. \tag{4}$$

Then, we can parallelize the decoding procedure using the following *generalized-sandwich* (GS) paradigm. It takes as the input the graph $(\mathcal{V}, \mathcal{E})$ and a set $D \subseteq \mathcal{V}$ of defects and outputs a set $K \subseteq \mathcal{E}$ of corrections, so that $\partial K = D$.

Algorithm 1 has several unspecified freedoms, such as the "partition method" and "inner decoder." In our work, the decoder graphs were partitioned along the time direction and the inner decoders took as input the buffer regions as well as the cores. It would be interesting to explore the many design choices for which valid corrections K_i can be found and yield low logical error rates.

Algorithm 1 provides one generalization of the sandwich decoder regarding disjoint core regions across windows (such as having non-negative seam offset). We can easily construct similar variants for overlapping core regions.

```
1: if $(V, E)$ consists of disconnected subgraphs, each of a small enough size then
2:     Apply the inner decoder to each subgraph in parallel, return the union of the outputs
3: Apply the partition method to choose "cores" $\{C_i \subseteq V\}_i$ with disjoint $\{\Delta(E, C_i)\}_i$, each of a small enough size
4: Apply the inner decoder in parallel to calculate corrections $K_i \subseteq \Delta(E, C_i)$, for all $i$, with $\partial K_i \cap C_i = D \cap C_i$
5: $V' \leftarrow V \setminus \bigsqcup_i C_i$ ,  $E' \leftarrow E \setminus \bigsqcup_i \Delta(E, C_i)$ ,  $D' \leftarrow D + \partial\left(\bigsqcup_i K_i\right)$
6: return $\mathrm{GS}(V', E', D') \sqcup \bigsqcup_i K_i$
```

Algorithm 1. Generalized sandwich decoder $\mathrm{GS}(V, E, D)$

VII. CONCLUSIONS

In this paper, we propose the sandwich sliding-window decoding scheme, which achieves high throughput by parallelizing overlapping windows in the time direction. To better motivate and illustrate our design principles, we reformulate previous decoding schemes as batch decoders and forward decoders and identify their limitations in parallelization and scalability.

We demonstrate with strong numerical evidence that our sandwich decoder has almost identical performance in accuracy and threshold compared to its batch-decoder counterparts. As for generalization, we provide a parallel divide-and-conquer formalism for our sandwich decoder, which applies to general stabilizer codes and possibly general logical operations. For example, for lattice surgery, we may need to divide syndromes into windows along the space direction, as well as the time direction. Further theoretical justification or more numerical validation is needed to fully evaluate our sandwich decoder in real-time decoding for fault-tolerant quantum computation.

ACKNOWLEDGMENTS

We would like to thank Xiaotong Ni for the insightful discussion and Hui-Hai Zhao for providing additional computational resource to speed-up the simulation. This work was supported by the Alibaba Group through the Alibaba Research Intern Program, and conducted when X.T. was a research intern at Alibaba Group USA.

Note added.—An independent study [41] has become available to the public concurrently with ours, addressing the same syndrome-backlog problem in surface-code decoding and sharing a similar methodology of dividing the decoder graph into windows. However, there are several differences, including the latency slow-down of the parallel decoder discussed in Ref. [41], which is not covered in our work. Our opinions also differ on the overlap of window divisions in subroutines, with our work lifting restrictions for a wider scope of application.

APPENDIX A: MONTE CARLO SIMULATION OF THE THRESHOLD

1. Union-find inner decoders

In most of our experiments, we use a UF decoder, as proposed in Ref. [33], to decode each individual window (including the forward windows and both types of the sandwich window). We use the *weighted-growth* version of the decoder described in Ref. [33, Sec. 5] but slightly modify the definition of the "boundary size" in our implementation.

We choose the UF decoder due to its low time complexity both in theory and in practice. However, it is unclear whether the UF decoder is the best fit for a sliding-window scheme. In theory, the UF decoder does not try to approximate the minimum-weight correction; instead, it tries to find an equivalence class that is likely to contain the actual error and then it chooses an arbitrary correction in that equivalence class with a simple peeling decoder. This means that the updated detectors at the right and left seams obtained by applying only part of the correction output by the UF decoder may be misleading, although our experimental results indicate that this does not noticeably affect the performance in practice.

In any case, we also conduct some experiments with a MWPM decoder as an alternative inner decoder to validate the universality of the sandwich scheme.

2. Sampling errors

We employ the circuit-level depolarizing error model with a single parameter p. More precisely, we assume that the preparation-and-measurement errors exist on all data and ancilla qubits, where a qubit is initialized to an orthogonal state with probability p and a measurement result is flipped with probability p. Each single-qubit, two-qubit, and idle gate is implemented as a perfect gate followed by a depolarizing channel. With probability p, the perfect gate is afflicted by a nontrivial Pauli fault chosen uniformly at random.

The faults attached to different elementary operations are applied independently.

3. Monte Carlo simulations

Given each code distance $d \in \{3, 5, \ldots, 17\}$, we choose a sandwich decoder with step size $s_d = (d+1)/2$ and window size $w_d = 3s_d$. We use the 3D MWPM and UF decoder, respectively, as the inner decoder to handle each type-1 window and the 2D MWPM and UF decoder, respectively, to handle each type-2 window (i.e., the seam offset is 0).

For the experiments with the UF inner decoder, we consider physical error rates $p \in \{0.3\%, 0.4\%, 0.5\%, 0.55\%, 0.6\%, 0.7\%, 0.8\%\}$; for the experiments with the MWPM inner decoder, we consider physical error rates $p \in \{0.4\%, 0.5\%, 0.6\%, 0.65\%, 0.7\%, 0.8\%\}$. For each p, we run a Monte Carlo simulation to find the logical error rate per d cycles for 100 000 shots. For the logical error rate plots for the sandwich and batch decoders using the UF and MWPM inner decoders, see Fig. 6.

To define the concept of the "logical error rate per d cycles," $p_{\mathrm{L}}(d)$, we make the ansatz that each cycle of syndrome extraction independently flips the logical qubit with a fixed probability $p_{\mathrm{L}}(1)$. If we exclude the data-qubit initialization and final measurement faults, the probability of flipping the logical qubit after i cycles of syndrome

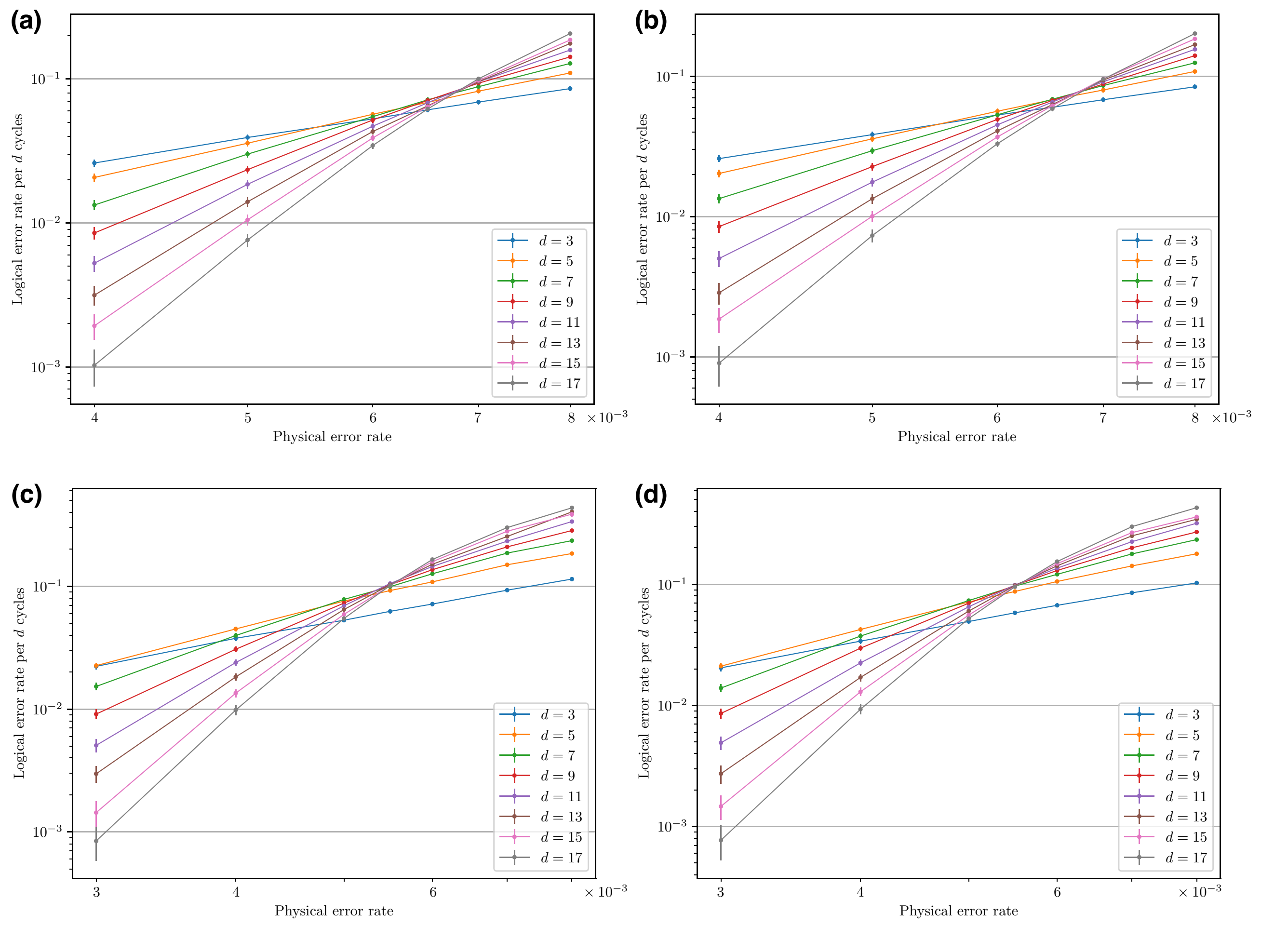

FIG. 6. The logical error rates for (a) the MWPM sandwich decoder, (b) the MWPM batch decoder, (c) the UF sandwich decoder, and (d) the UF batch decoder for code distances $d = 3, 5, \ldots, 17$. The logical error rates per d cycles $p_{\mathrm{L}}(d)$ are estimated with weighted least squares, as explained in Sec. A 3. The error bars indicate 95% statistical confidence according to a conservative estimate (i.e., an overestimate) of the variance of $\hat{p}_{\mathrm{L}}(d)$, as explained in Sec. A 4. The sandwich decoder has almost the same performance as the corresponding batch decoder.

extraction $p_{\mathrm{L}}(i)$ satisfies

$$1 - 2p_{\mathrm{L}}(i) = [1 - 2p_{\mathrm{L}}(1)]^i. \tag{A1}$$

Define q to be the probability that the data-qubit initialization and measurement collectively flip the logical qubit. Then, the probability $p_{\mathrm{L},n}$ of logical error for an n-cycle memory experiment satisfies

$$1 - 2p_{\mathrm{L},n} = (1 - 2q) \cdot (1 - 2p_{\mathrm{L}}(1))^n. \tag{A2}$$

We can thus calculate the logical error rate per d cycles $p_{\mathrm{L}}(d)$ from the estimated logical error rate per shot $\hat{p}_{\mathrm{L},n}$ using the weighted least-squares estimator

$$\log(1 - 2\hat{p}_{\mathrm{L}}(d)) = \mathbf{x}^\top \mathrm{diag}(\mathbf{w})\, \mathbf{y} \Big/ \mathbf{x}^\top \mathrm{diag}(\mathbf{w})\, \mathbf{x}, \tag{A3}$$

where

$$\begin{aligned} x_n &= (n - \overline{n})/d, \\ w_n &= 1/\hat{\mathrm{Var}}(y_n), && \text{(A4)} \\ y_n &= \log(1 - 2\hat{p}_{\mathrm{L},n}) - \overline{\log(1 - 2\hat{p}_{\mathrm{L},n})}. && \text{(A5)} \end{aligned}$$

The explicit form of the estimator $\hat{\mathrm{Var}}(y_n)$ will be given in Sec. A 4. In our experiments, we simulate with different overall numbers of cycles $n = \lfloor k s_d/2 \rfloor$, where $k \in \{8, 9, \ldots, 20\}$.

To more efficiently simulate the behavior of our scheme for different numbers of cycles, we conduct those simulation experiments simultaneously, reusing the sampled errors and decoder outputs for early cycles. That is, for each d and p, we only construct one decoder graph with $n = 10s_d$ and sample errors on it. Then, within the same decoder graph, we calculate the logical error rates per shot

TABLE I. The fitting parameters for the ansatz in Eq. (A9) and their standard deviations.

	p_{th}	A	B	C	D	ν	μ
MWPM sandwich	$6.80(2) \times 10^{-3}$	$8.4(1) \times 10^{-2}$	4.8(8)	$7(2) \times 10^{1}$	$-6.6(0)$	1.00(6)	$2(0) \times 10^{-2}$
UF sandwich	$5.54(1) \times 10^{-3}$	$1.06(1) \times 10^{-1}$	6(1)	$1.0(3) \times 10^{2}$	-1.1×10^{1}	1.00(6)	$2(0) \times 10^{-2}$
MWPM batch	$6.83(2) \times 10^{-3}$	$8.2(1) \times 10^{-2}$	5.3(8)	$1.0(3) \times 10^{2}$	8.9(0)	1.05(6)	$2(0) \times 10^{-2}$
UF batch	$5.53(2) \times 10^{-3}$	$9.9(2) \times 10^{-2}$	6(1)	$1.0(3) \times 10^{2}$	-4.9×10^{1}	1.00(6)	$2(0) \times 10^{-2}$

for all $n = \lfloor ks_d/2 \rfloor$, where $k \in \{8, 9, \ldots, 20\}$. This causes the results of those experiments to be correlated but over the 100 000 independent shots, the effect of this correlation should be minor.

More specifically, each simulation proceeds as follows:

(1) Before starting any sliding, sample edges on the entire decoder graph according to precalculated probabilities, except for the final data-qubit measurement errors.

(2) Decode each type-1 window with defects generated from the sampled edges and identify the core region. More specifically:

(a) For the last layer in the decoder graph, the simulated errors are evaluated from the sampled data-qubit measurement errors [recall Eq. (1)].

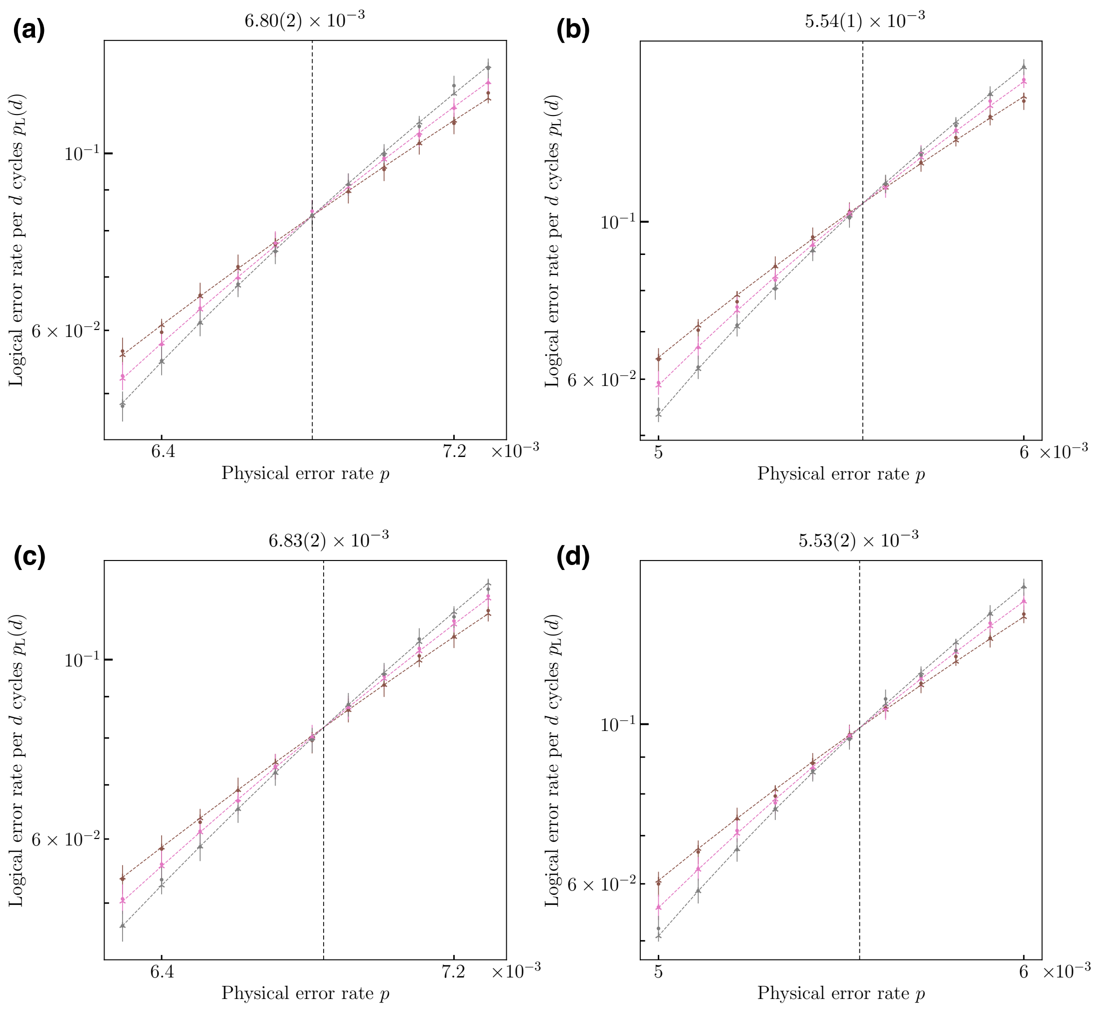

FIG. 7. The fitting curves of the logical error rates of the (a) MWPM sandwich, (b) UF sandwich, (c) MWPM batch, and (d) UF batch decoders. Here, $d = 13, 15, 17$ and p increases from 0.0063 to 0.0073 for MWPM and from 0.005 to 0.006 for UF, both in increments of 0.0001. The vertical lines indicate thresholds and the colored lines are the fitting curves, all using parameters in Table I.

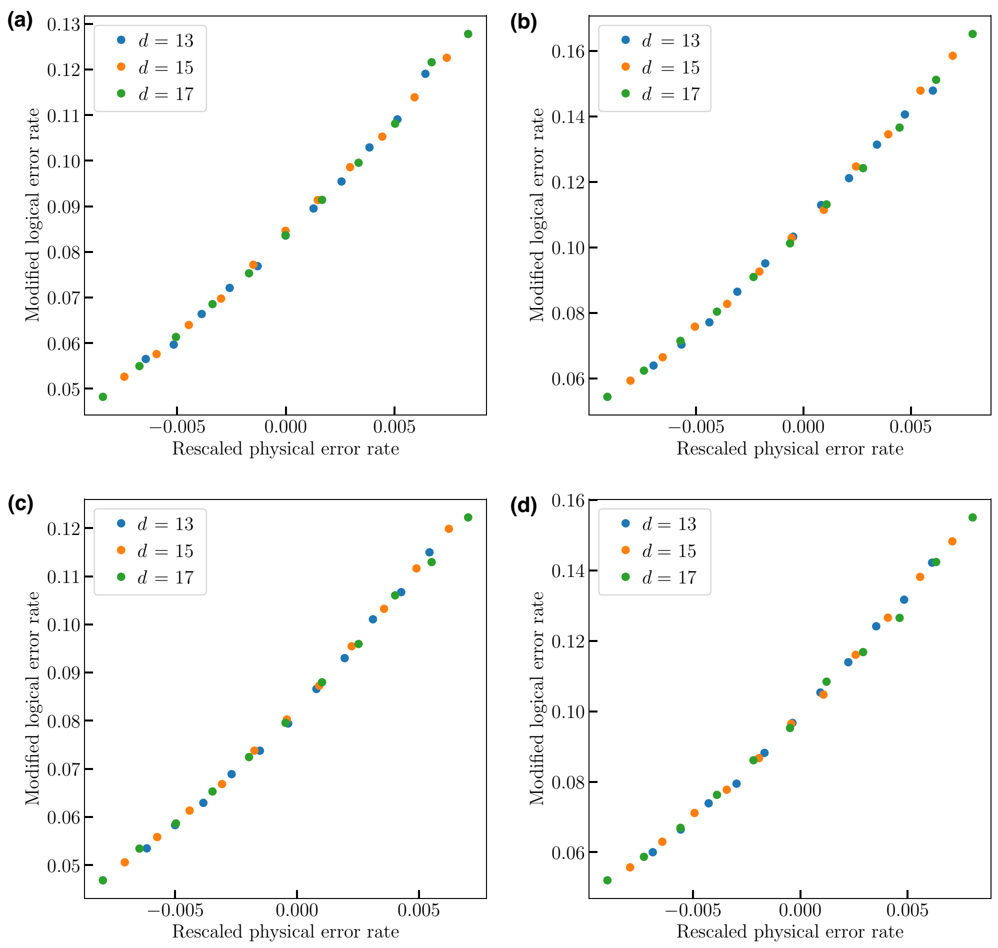

FIG. 8. Rescaled logical error rates for the (a) MWPM sandwich, (b) UF sandwich, (c) MWPM batch, and (d) UF batch decoders. Here, $d = 13, 15, 17$ and p increases from 0.0063 to 0.0073 for MWPM and from 0.005 to 0.006 for UF, both in increments of 0.0001. The vertical axis is $A + Bx + Cx^2 = p_{\mathrm{L}}(d) - Dd^{-1/\mu}$ and the horizontal axis is $x = (p - p_{\mathrm{th}})d^{1/\nu}$, all using parameters in Table I.

Then, decode this final window and apply corrections in the core region. Finally, calculate the logical error rate per d cycles.

(b) When the window has not reached the last cycle but would be the final window if the number of cycles were to equal $\lfloor ks_d/2 \rfloor$ for $k \in \{8, 9, \ldots, 19\}$, make two copies of the current simulations. On one copy, treat this window as the final window in case (a). On the other copy, proceed as in case (c).

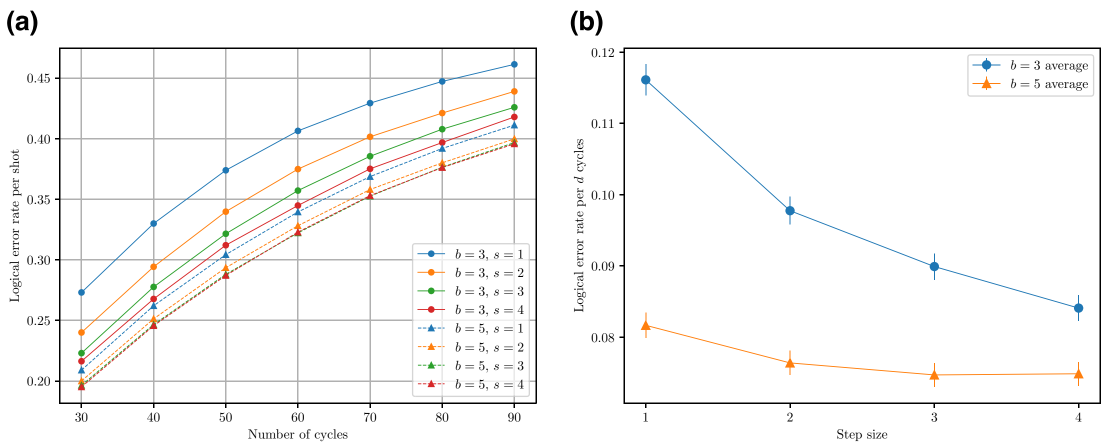

FIG. 9. The basic analysis on step size s and buffer size b when $d = 9$, $p = 0.005$, and the numbers of cycles $n \in \{30, 40, 50, 60, 70, 80, 90\}$. Recall that the sandwich-window size w is $s + 2b$. We fix the buffer size to 3 and 5 and vary the step size. (a) The x axis is the number of cycles and the y axis is the logical error rate per shot. (b) Converting logical error rates per shot to logical error rates per d cycles and changing the x axis to the step size.

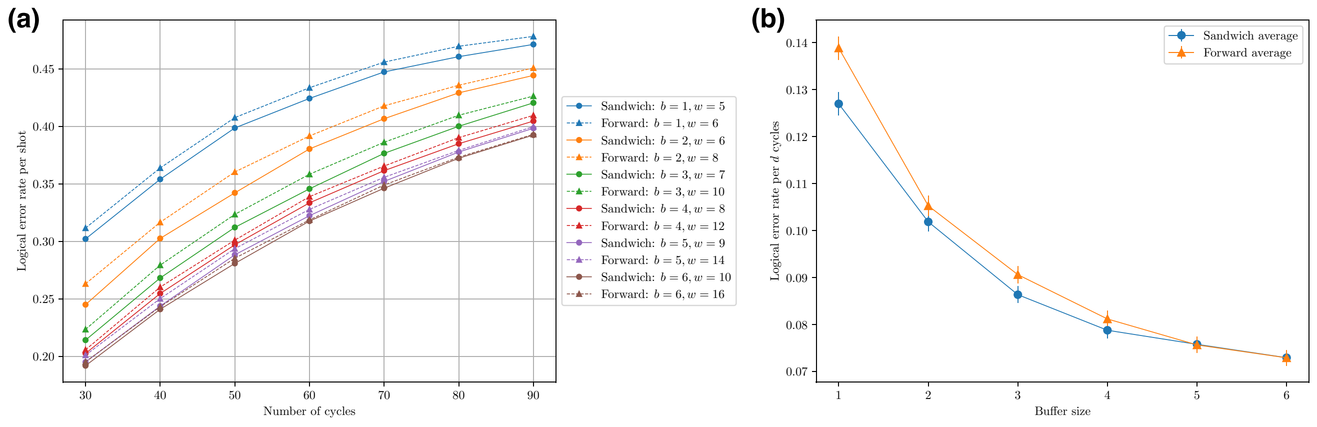

FIG. 10. A comparison between forward windows and sandwich windows. We fix $d=9$, $p=0.005$, and the step size $s=4$. The forward-window size is $s+b$ and the sandwich-window size is $s+2b$.

(c) When the window is not the final window for any numbers of cycles requested, decode, apply correction in the core region, and slide to the next window.

(3) For each type-2 window (i.e., the overlapping seam between two consecutive decoded type-1 windows), put all inconsistent detectors into a 2D decoder and apply all the generated corrections.

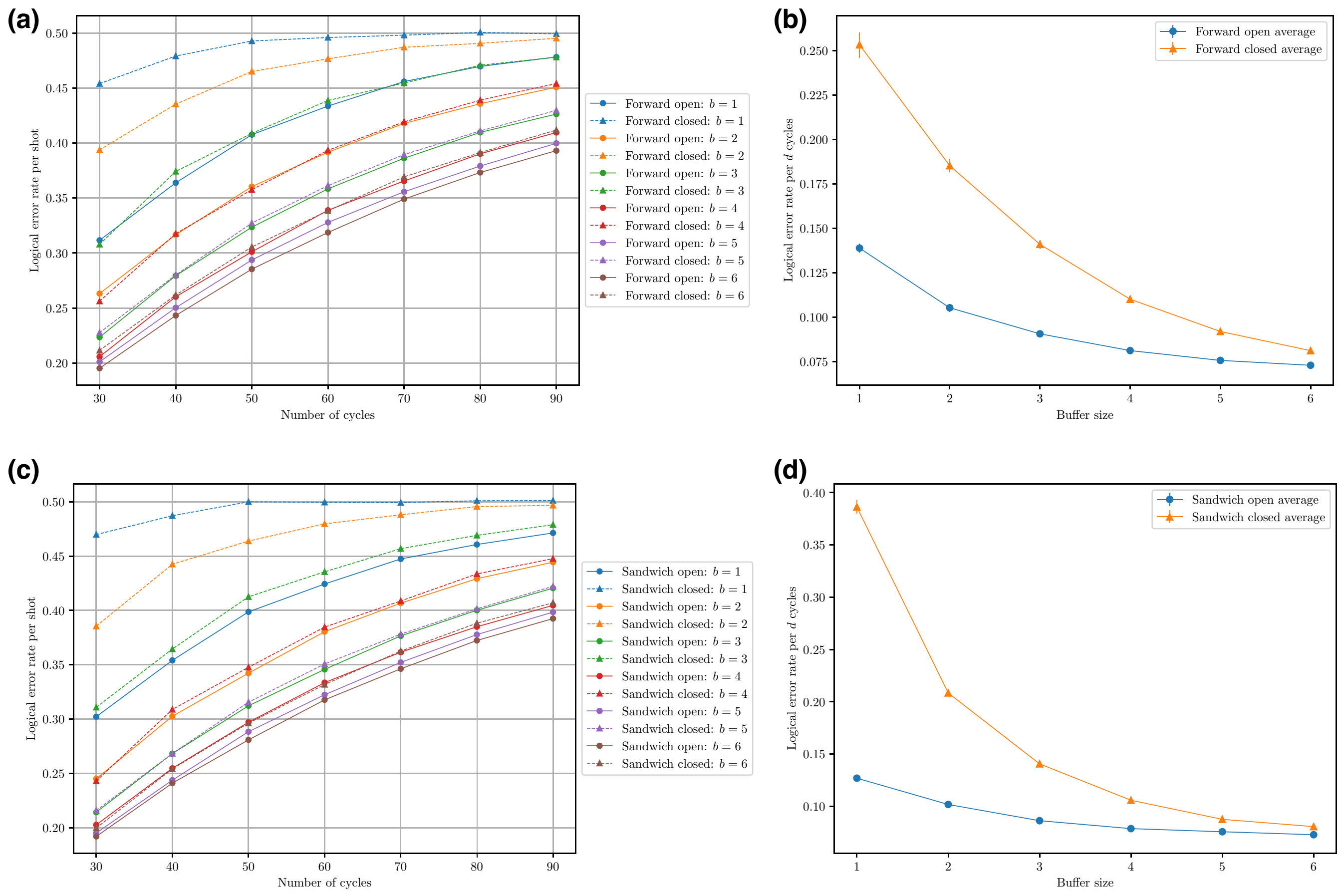

FIG. 11. A comparison between open and closed artificial time boundaries for forward windows. We fix $d=9$, $p=0.005$, and $s=4$.

4. Error estimation

For each value of n (and combination of other parameters), our Monte Carlo simulation gives an estimated logical error rate per shot $\hat{p}_{\mathrm{L},n}$ with variance

$$\mathrm{Var}(\hat{p}_{\mathrm{L},n}) = \frac{p_{\mathrm{L},n} \cdot (1 - p_{\mathrm{L},n})}{N}, \tag{A6}$$

where $N = 10^5$ denotes the number of shots. The weights w_i used in the least-squares estimator given in Eq. (A3) are derived from the approximate variance,

$$\begin{aligned}\hat{\mathrm{Var}}(y_n) &= \hat{\mathrm{Var}}\left(\log(1 - 2\hat{p}_{\mathrm{L},n})\right) \\ &\approx \left(\frac{2}{2\hat{p}_{\mathrm{L},n} - 1}\right)^2 \cdot \hat{\mathrm{Var}}(\hat{p}_{\mathrm{L},n}).\end{aligned} \tag{A7}$$

As mentioned above, our estimations $\hat{p}_{\mathrm{L},n}$ for different values of n are correlated. Therefore, we cannot use the usual variance estimator for weighted least squares. Instead, we take a conservative estimate of the variance:

$$\begin{aligned}&\hat{\mathrm{Var}}\left(\log(1 - 2\hat{p}_{\mathrm{L}}(d))\right) \\ &\lessapprox \left(\sum_n |x_n| w_n \sqrt{\hat{\mathrm{Var}}(y_n)} \Big/ \sum_n w_n x_n^2\right)^2.\end{aligned} \tag{A8}$$

5. Thresholds

We evaluate the thresholds for the sandwich and batch decoders with the MWPM and UF inner decoders by curve fitting with the same ansatz as Eq. (43) in Ref. [42]. Specifically,

$$\begin{aligned}p_{\mathrm{L}}(d) &= A + Bx + Cx^2 + D\, d^{-1/\mu}, \\ x &= (p - p_{\mathrm{th}})\, d^{1/\nu}.\end{aligned} \tag{A9}$$

For all the four combinations of decoding schemes and inner decoders, we perform Monte Carlo simulations as in Sec. A 3, with $d = 13, 15, 17$ and p increasing from 0.005 to 0.006 for UF and from 0.0063 to 0.0073 for MWPM, both in increments of 0.0001. For the fitting parameters, see Table I and Figs. 7 and 8.

APPENDIX B: NUMERICAL ANALYSIS

1. Step size and window size

When we evaluate the performance of sandwich decoders, the step size s and window size w are two natural features to consider:

$$w = s + 2 \times \text{buffer size } (b). \tag{B1}$$

As the number of cycles increases, the logical error rate per shot also increases and gradually converges to 0.5, following from Eq. (A2). Figure 9 gives an example for $d = 9$ and $p = 0.005$. Generally speaking, a larger buffer size, as well as a larger step size when the buffer size is fixed, results in lower logical error rates.

2. Sandwich versus forward windows

During each sandwich window, since we only accept the corrections in the middle, the decoder receives defect information from both the future and the past. The forward window [5,30] can prevent premature matchings by taking account into its most recent future detection events. However, it sometimes fails to prevent problematic matchings from the past because the knowledge that the decoder holds of its most recent past events is limited to a single layer of propagated syndromes.

Indeed, we show in Fig. 10 that when we release the same amount of future events to both the sandwich window and the forward window, i.e., fix the step size and the size of the buffer region(s), the sandwich windows produce lower logical error rates. Moreover, it means that given any forward decoder, we can construct a sandwich decoder such that it performs comparatively well or even better.

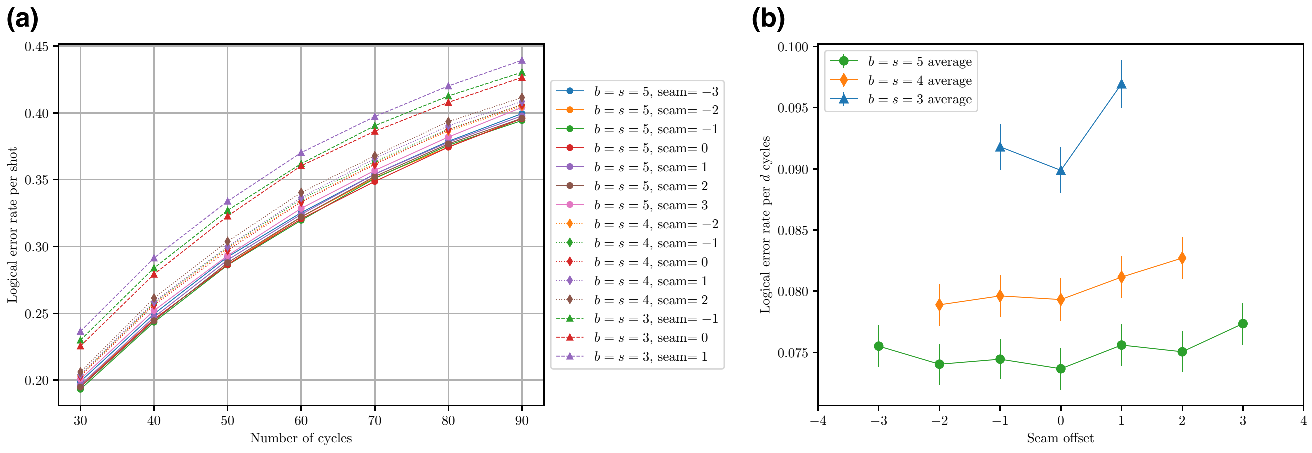

FIG. 12. A comparison between different seam offsets for sandwich decoders. We fix $d = 9$ and $p = 0.005$.

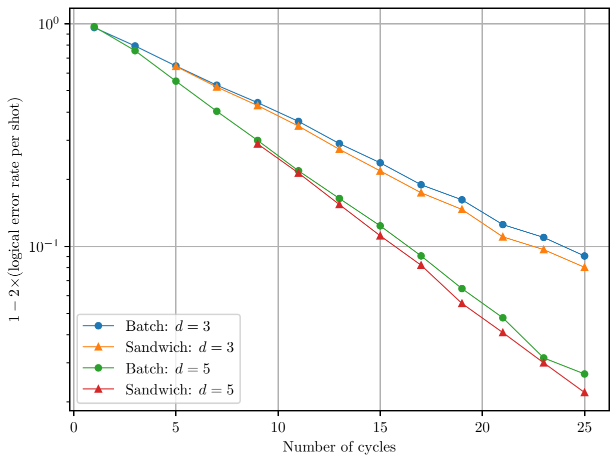

FIG. 13. A comparison between the UF sandwich decoder and the UF batch decoder on Google QEC data. For $d=3$, each point is the average of four configurations of the surface codes as specified in Ref. [40]. There is only one configuration when $d=5$. Each configuration has 50 000 shots. The sandwich decoder that we choose has a step size of $(d+1)/2$ and a window size of $3(d+1)/2$. We plot in a style similar to that of Ref. [40, Fig. 3], where the y axis is the logical fidelity $1-2p_{\mathrm{L}}(n)$ and n is the number of cycles. If follows from Eq. (A2) that the slope indicates the logical error rate per cycle.

3. Open versus closed time boundaries

As mentioned in Sec. III D, it is unclear in Ref. [30] whether the future boundaries of windows are generally open or closed. Therefore, we do a series of simulation experiments where we close all the time boundaries in windows and compare the performance with the normal case where only the past (left) time boundary of the first window and the future (right) time boundary of the last window are closed. As shown in Fig. 11, open boundaries lead to lower logical error rates compared to closed boundaries. As we increase the buffer size, the advantage becomes less obvious.

4. Seam offset for inconsistent corrections

In Fig. 12, we study the effect of having different seam offsets (Fig. 3). The results seem to vaguely indicate that our choice of seam offset 0 is actually the best or close to the best value, although the difference is small and the data are far from conclusive.

5. Real-world data from Google QEC experiments

We also evaluate the performance of the sandwich (UF) decoders on real-world data provided in Ref. [40] for $d=3$ and $d=5$. As shown in Fig. 13, the sandwich decoders only have slightly larger logical error rates compared with those obtained by the batch (UF) decoders. Note that Ref. [40] used tensor-network and belief decoders, which inherently have higher accuracy than UF decoders, to obtain the suppression of errors from $d=3$ to $d=5$.

[1] D. Aharonov and M. Ben-Or, Fault-tolerant quantum computation with constant error rate, SIAM J. Comput. **38**, 1207 (2008).
[2] E. Knill, Quantum computing with realistically noisy devices, Nature **434**, 39 (2005).
[3] P. Aliferis, D. Gottesman, and J. Preskill, Quantum accuracy threshold for concatenated distance-3 code, Quantum Inf. Comput. **6**, 97 (2006).
[4] D. Gottesman, Fault-tolerant quantum computation with constant overhead, Quantum Inf. Comput. **14**, 1338 (2014).
[5] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, J. Math. Phys. **43**, 4452 (2002).
[6] 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).
[7] A. Kitaev, Fault-tolerant quantum computation by anyons, Ann. Phys. **303**, 2 (2003).
[8] S. Bravyi and A. Kitaev, Universal quantum computation with ideal Clifford gates and noisy ancillas, Phys. Rev. A **71**, 022316 (2005).
[9] B. M. Terhal, Quantum error correction for quantum memories, Rev. Mod. Phys. **87**, 307 (2015).
[10] R. Raussendorf and J. Harrington, Fault-tolerant quantum computation with high threshold in two dimensions, Phys. Rev. Lett. **98**, 190504 (2007).
[11] A. G. Fowler, A. M. Stephens, and P. Groszkowski, High-threshold universal quantum computation on the surface code, Phys. Rev. A **80**, 052312 (2009).
[12] J. W. Harrington, Ph.D. thesis, The Division of Physics, Mathematics and Astronomy, California Institute of Technology, 2004.
[13] G. Duclos-Cianci and D. Poulin, Fast decoders for topological quantum codes, Phys. Rev. Lett. **104**, 050504 (2010).
[14] S. Bravyi and J. Haah, Quantum self-correction in the 3D cubic code model, Phys. Rev. Lett. **111**, 200501 (2013).
[15] K. Fujii, M. Negoro, N. Imoto, and M. Kitagawa, Measurement-free topological protection using dissipative feedback, Phys. Rev. X **4**, 041039 (2014).
[16] M. Herold, E. T. Campbell, J. Eisert, and M. J. Kastoryano, Cellular-automaton decoders for topological quantum memories, npj Quantum Inf. **1**, 1 (2015).
[17] G. Torlai and R. G. Melko, Neural decoder for topological codes, Phys. Rev. Lett. **119**, 030501 (2017).
[18] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong, in *Proc. 47th ACM/IEEE Int. Symp. C. (ISCA)* (IEEE Press, 2020), p. 556.
[19] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, in *2022 IEEE Int. S. High Perf. Comp. (HPCA)* (IEEE, 2022), p. 274.
[20] N. Delfosse, Hierarchical decoding to reduce hardware requirements for quantum computing, arXiv:2001.11427 (2020).
[21] K. Meinerz, C.-Y. Park, and S. Trebst, Scalable neural decoder for topological surface codes, Phys. Rev. Lett. **128**, 080505 (2022).

[22] S. Gicev, L. C. L. Hollenberg, and M. Usman, A scalable and fast artificial neural network syndrome decoder for surface codes, Quantum **7**, 1058 (2023).

[23] C. Chamberland, L. Goncalves, P. Sivarajah, E. Peterson, and S. Grimberg, Techniques for combining fast local decoders with global decoders under circuit-level noise, Quantum Sci. Technol. **8**, 045011 (2023).

[24] S. C. Smith, B. J. Brown, and S. D. Bartlett, Local predecoder to reduce the bandwidth and latency of quantum error correction, Phys. Rev. Appl. **19**, 034050 (2023).

[25] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, NEO-QEC: Neural network enhanced online superconducting decoder for surface codes, arXiv:2208.05758 (2022).

[26] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, Towards practical classical processing for the surface code, Phys. Rev. Lett. **108**, 180501 (2012).

[27] A. G. Fowler, Minimum weight perfect matching of fault-tolerant topological quantum error correction in average O(1) parallel time, Quantum Inf. Comput. **15**, 145 (2015).

[28] P. Das, C. A. Pattison, S. Manne, D. M. Carmean, K. M. Svore, M. Qureshi, and N. Delfosse, in *2022 IEEE Int. S. High Perf. Comp. (HPCA)* (IEEE, 2022), p. 259.

[29] R. J. Overwater, M. Babaie, and F. Sebastiano, Neural-network decoders for quantum error correction using surface codes: A space exploration of the hardware cost-performance tradeoffs, IEEE Trans. Quantum Eng. **3**, 1 (2022).

[30] P. Das, A. Locharla, and C. Jones, in *Proc. 27th ACM ASPLOS* (ACM, New York, NY, USA, 2022), p. 541.

[31] S. Bartolucci, P. Birchall, H. Bombín, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant, F. Pastawski, T. Rudolph, and C. Sparrow, Fusion-based quantum computation, Nat. Commun. **14**, 912 (2023).

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

[33] N. Delfosse and N. H. Nickerson, Almost-linear time decoding algorithm for topological codes, Quantum **5**, 595 (2021).

[34] When a fault flips only one real detector on an intersection of space and time boundaries, it is impossible to distinguish if the fault afflicts, e.g., an ancilla measurement or a boundary data qubit. To conceptually simplify the decoding and its simulation, however, in Fig. 1(c) we connect such open real detectors to two imaginary detectors rather than one.

[35] Hereafter, whenever we say “top,” “bottom,” “left,” or “right” in the context of space boundaries, we refer to the directions in Fig. 1(a).

[36] The longest series of sequential operations in a parallel computation.

[37] Technically, we still need to add up the logical corrections from all the windows but this has negligible computational cost in practice and theoretically it can be done in $O(\log n)$ parallel time.

[38] C. Gidney, Stability experiments: The overlooked dual of memory experiments, Quantum **6**, 786 (2022).

[39] C. Chamberland and E. T. Campbell, Universal quantum computing with twist-free and temporally encoded lattice surgery, PRX Quantum **3**, 010331 (2022).

[40] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, and R. Babbush *et al.*, Suppressing quantum errors by scaling a surface code logical qubit, Nature **614**, 676 (2023).

[41] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell, Parallel window decoding enables scalable fault tolerant quantum computation, Nat. Commun. **14**, 7040 (2023).

[42] C. Wang, J. Harrington, and J. Preskill, Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory, Ann. Phys. **303**, 31 (2003).