Fast and accurate AI-based pre-decoders for color codes

Jan Olle,[1] Christopher Chamberland,[1,*] Muyuan Li,[1] and Igor Baratta[1]
[1] *NVIDIA Corporation, USA*

Color codes are promising alternatives to surface codes for universal fault-tolerant quantum computing due to their simpler lattice-surgery protocols and the transversal implementation of logical Clifford gates. However, their practical deployment has been limited by slower decoding algorithms and worse logical failure rates and thresholds compared to surface codes. Although AI-based logical-flip decoders have recently been proposed to address these challenges, no clear framework currently exists for implementing such decoders within the parallel block-wise decoding schemes in both space and time required for large-scale fault-tolerant computation. AI-based pre-decoders offer a scalable alternative due to their local nature. By performing spacelike corrections on physical qubits and timelike corrections on stabilizer measurements, pre-decoders are naturally compatible with parallel block-wise decoding schemes and lattice-surgery protocols. In this work, we introduce AI-based pre-decoders for triangular color codes. We present a novel neural-network architecture for their implementation and develop methods to simplify the complex training data generated by color-code syndrome-extraction circuits containing feedforward operations. Remarkably, we find that both logical failure rates (LERs) and runtimes improve relative to raw Chromobius decoding as the code distance increases. For example, at code distance d=31 and physical error rate $p = 0.3\%$, our pre-decoder + Chromobius pipeline improves the logical failure rate by a factor of 347x while reducing runtime by 7.33x compared to raw Chromobius decoding alone. These results demonstrate that AI-based pre-decoding can substantially narrow the performance gap between color codes and surface codes, bringing color codes closer to practical large-scale fault-tolerant quantum computation.

I. INTRODUCTION

Fault-tolerant quantum error correction will be essential for the implementation of large-scale quantum algorithms. Numerous quantum error-correcting codes have been proposed as candidates for universal fault-tolerant quantum computation on realistic hardware platforms. Among the most prominent are planar surface codes [1–3] and color codes [4–7]. Both are two-dimensional topological codes whose stabilizers can be measured using only nearest-neighbor interactions. Moreover, universal fault-tolerant logic can be implemented using nearest-neighbor operations when lattice surgery [8–13] is combined with magic-state distillation protocols [11, 14–19].

Color codes possess several advantages over surface codes for fault-tolerant logic. Most notably, all logical Clifford gates admit transversal implementations, substantially simplifying logical gate constructions. In addition, logical Y-measurements can be implemented without the use of twist defects and the asymmetric routing-space corridor configurations required in surface-code architectures.

Despite these advantages, surface codes remain the dominant architecture currently considered for large-scale fault-tolerant quantum computation. Two primary factors contribute to this. First, color codes are significantly more difficult to decode. State-of-the-art color-code decoders such as Chromobius [7] are generally slower than Minimum-Weight Perfect Matching (MWPM) [20, 21] and Union-Find (UF) [22] decoders commonly used for surface codes. This makes real-time decoding more challenging, particularly for superconducting hardware architectures where decoding latencies approaching 1μs per syndrome round may be required to avoid the exponential backlog problem [23–26]. Second, color codes typically require more physical qubits than surface codes to achieve the same target logical error rate (LER), resulting in larger resource overheads.

Recently, several new decoding approaches have been proposed to address these limitations. Ref. [27] introduced the VibeLSD decoder, which achieves target LERs using a number of physical qubits comparable to that of surface codes. However, because the algorithm relies on serialized belief propagation, further work is needed to demonstrate decoding runtimes compatible with real-time operation at large code distances. AI-based decoders for color codes such as AlphaQubit 2 [28] achieve near-optimal LER performance for both surface and color codes when provided with sufficient training data and optimization time. However, it remains unclear how such logical-classifier decoders can be integrated into the parallel block-wise decoding schemes in both space and time required for large-scale lattice-surgery computations [25, 26].

In lattice-surgery protocols, logical operations often produce extremely large merged code patches whose associated space-time decoding volumes are too large to decode monolithically in real time. Parallel block-wise decoding methods address this challenge by partitioning the global decoding volume into commit regions, associated buffer regions, and cleanup regions. To resolve residual syndromes within cleanup regions, physical spacelike and timelike corrections must be applied within commit regions. Since logical-classifier decoders infer only the

* Corresponding author; cchamberland@nvidia.com

values of logical observables rather than physical corrections throughout the decoding volume, it is unclear how they can be directly adapted to resolve cleanup-region syndromes within such frameworks.

In this work, we propose an AI-based pre-decoder for triangular color codes. Pre-decoders [24, 29–31] are trained to apply local corrections using only local spatio-temporal information. They output both spacelike and timelike physical corrections across all syndrome-measurement rounds and can be trained on a fixed volume $d_x \times d_z \times d_m$ while generalizing to decoding volumes of arbitrary size $d'_x \times d'_z \times d'_m$. Because pre-decoders are local and probabilistic, residual errors generally remain after correction and must therefore be resolved using a global decoder. However, since the runtime of most global decoders depends strongly on syndrome density, pre-decoders can substantially accelerate global decoding by sparsifying the residual syndrome. If the pre-decoder latency is sufficiently small, the combined pre-decoder + global decoder pipeline can achieve lower total runtime than the standalone global decoder. In many cases, such pipelines also improve logical error rates relative to the global decoder alone. Most importantly, because pre-decoders generate physical spacelike and timelike corrections throughout the decoding volume, they are naturally compatible with parallel block-wise decoding schemes in both space and time.

In this work, we use Chromobius [7] as the global decoder due to its strong combination of decoding performance and runtime efficiency. Although VibeLSD achieves lower LERs than Chromobius, competitive real-time decoding runtimes have not yet been demonstrated. We show that our pre-decoder + Chromobius pipeline achieves both substantial LER reductions and runtime improvements relative to standalone Chromobius decoding. Moreover, these improvements become increasingly pronounced as the code distance grows and persist both above and below the physically relevant regime of $p = 0.1\%$. For example, at $d = 31$ and $p = 0.3\%$, our pipeline achieves a 347x reduction in LER together with a 7.33x runtime improvement relative to Chromobius alone.

This paper is organized as follows. In Section II, we briefly review the fundamentals of color codes. In Section III, we describe the neural-network architecture used for our AI-based pre-decoders and introduce data-processing techniques that substantially improve training performance. In Section IV, we present detailed numerical results for both LER and runtime improvements obtained using the pre-decoder + Chromobius pipeline. Finally, we conclude and discuss future directions in Section V.

II. BRIEF REVIEW OF THE COLOR CODE

In this section, we review the key properties of triangular color codes relevant to the pre-decoder architecture described in Section III.

Triangular color codes are obtained by cutting a tiling of hexagons with an equilateral triangle, resulting in the lattice shown in Fig. 1a. The resulting lattice satisfies the 3-valence condition: all vertices, except for the three corner vertices, are incident to exactly three edges. The lattice is also 3-colorable, meaning that each plaquette can be assigned one of three colors (e.g., red, green, and blue) such that neighboring plaquettes sharing an edge have different colors [5, 6, 32].

Each plaquette supports both an X-type and a Z-type stabilizer acting on the data qubits located at its vertices (shown as yellow vertices in Fig. 1a). For each Pauli type $P \in \{X, Z\}$, the logical operator P_L can be represented by a string of identical Pauli operators supported along any one of the three boundaries of the triangular lattice.

In this work, we use the "superdense" syndrome-extraction circuits introduced in Ref. [7] and shown in Figs. 1b and 1c to measure the weight-6 and weight-4 stabilizers of the color code. To maintain a regular lattice geometry for both data qubits and ancillas, we use the modified boundary circuits shown in Figs. 1d and 1e for the red and green weight-4 stabilizers.

A key advantage of the circuits in Fig. 1 is that the X- and Z-type stabilizers can be measured simultaneously. Specifically, the ancilla prepared in $|+\rangle$ ($|0\rangle$) encodes the X-type (Z-type) stabilizer measurement outcome. Similar circuits were previously considered in Ref. [33], where the second ancilla was instead used as a flag qubit [17, 18, 34–36]. However, that approach requires repeating the circuit twice in order to measure both stabilizers, resulting in a larger number of fault locations.

Recently, Ref. [7] introduced Chromobius, an open-source implementation of the möbius color-code decoder. Chromobius relies on Minimum-Weight Perfect Matching (MWPM) [20, 21] applied to subgraphs of the decoding problem, and consequently its runtime depends strongly on the syndrome density (i.e., the number of detection events). As such, Chromobius is particularly well suited for use within an AI-based pre-decoding pipeline [24, 29–31], where the pre-decoder reduces the syndrome density prior to global decoding and thereby simplifies the downstream matching problem. Nevertheless, we emphasize that any global decoder can be used in conjunction with our AI-based pre-decoder.

III. NEURAL NETWORK ARCHITECTURE AND HYPERPARAMETERS

In this section, we present the AI-based pre-decoder architecture used for triangular color codes. Many of the techniques introduced here are adapted from Ref. [31] and generalized to the color-code setting. An overview of the full decoding pipeline incorporating a pre-decoder is shown in Fig. 2.

At a high level, the pre-decoder receives syndrome data from the quantum processing unit (QPU) and applies local spacelike and timelike corrections. Because AI-based pre-decoders are probabilistic and local in nature, residual

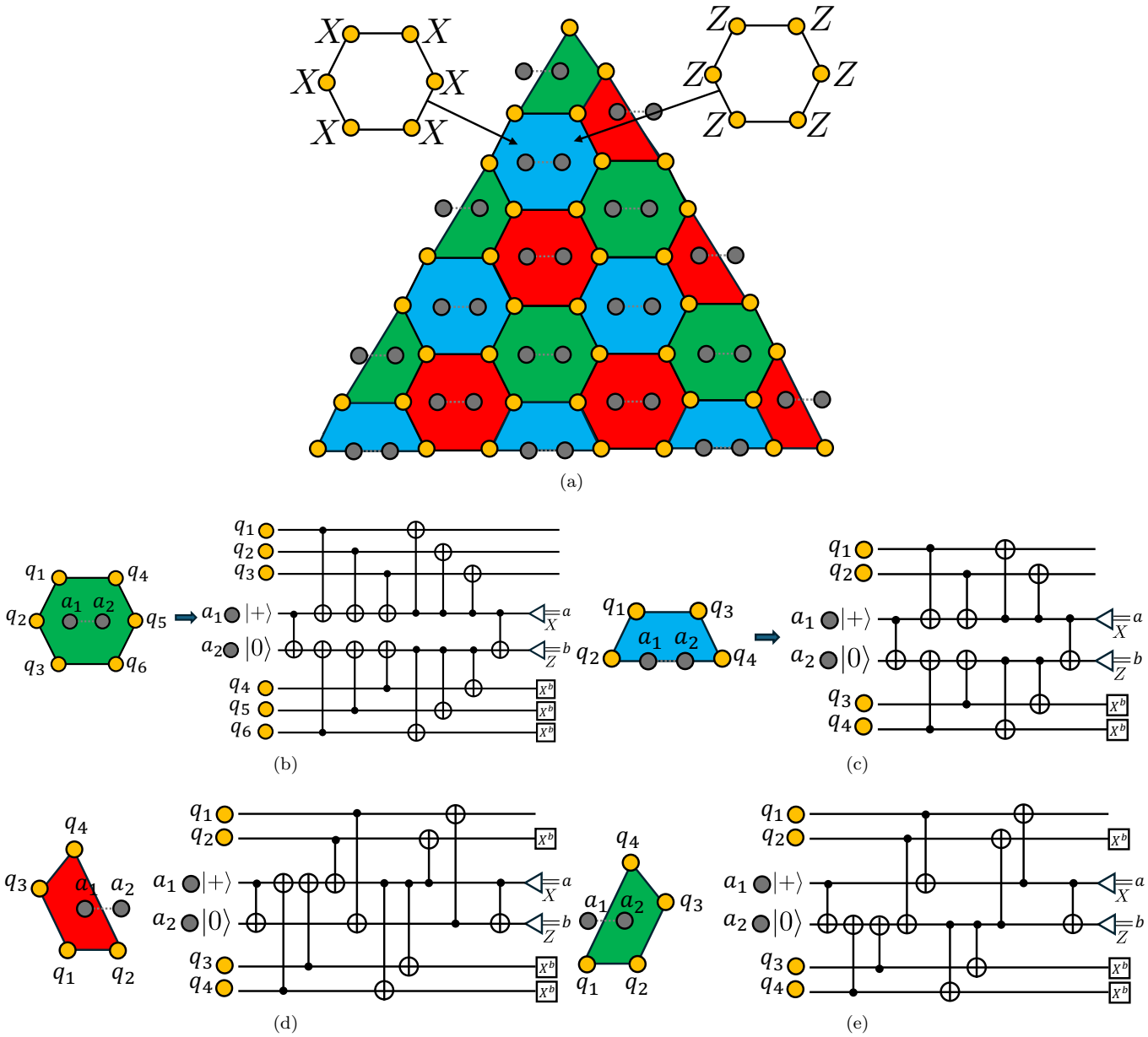
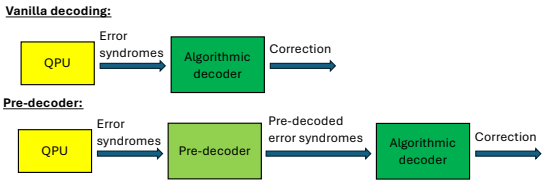


FIG. 1. (a) A $d = 7$ lattice for the triangular color code. Each weight-4 and weight-6 plaquette corresponds to both an X and Z-type stabilizer. Data qubits shown by yellow vertices, with gray vertices corresponding to the ancillas used to measure the stabilizers. Logical representatives for X_L, Z_L and Y_L operators correspond to tensor product of X, Z and Y Paulis on data qubits along any of the boundaries of the triangle. (b) Circuit to measure both weight-6 X and Z-type stabilizers. (c) Same as (b) but for weight-4 stabilizers. In (d) and (e), we show the circuits used for red and green weight-four stabilizers which maintains the regular grid tilling of all qubits.

errors may remain after correction. Processed syndromes are therefore computed from the residual detection events together with the current values of the relevant logical observables. These processed syndromes are subsequently passed to a global decoder, which computes the final logical-observable corrections. In this work, Chromobius is used as the global decoder.

In Section III A, we review the fully convolutional neural-network architecture used for pre-decoding and introduce a mapping from the triangular color-code lattice onto a rectangular grid. Using this mapping, we describe how the input data used for both training and inference is generated. In Section III B, we explain how spacelike and timelike failures are labeled within the grid representation. We additionally review data-processing techniques, first introduced in Ref. [31], which substantially reduce the number of residual timelike failures following pre-decoding.

In Section III C, we describe an important simulation procedure used during training-data generation to elimi-



(a)

FIG. 2. Comparison between a vanilla decoding algorithm and a decoding algorithm that uses a pre-decoder to pre-process the error syndromes. When using a pre-decoder, the pre-decoder receives the error syndrome from the QPU and applies spacelike and timelike corrections across all syndrome measurement rounds that were used as inputs. The new error syndrome obtained from the corrections are then passed to an algorithmic decoder to apply the final set of corrections. Figure taken from Ref. [31].

nate artificial X-type error asymmetries arising from the feedforward operations present in the syndrome-extraction circuits of Fig. 1. Finally, in Section III D, we introduce homological-equivalence transformations tailored to triangular color codes that add additional structure to the labeled training data. We find that the data processing methods of Sections III C and III D significantly improve pre-decoder performance.

A. Input training data and grid mapping

In this work we consider two architectures for our AI-based pre-decoders. The first uses a fully convolutional neural-network architecture with three-dimensional convolutions which allows our networks to be applicable to input volumes which are of different size than the one it was trained on. An illustration of the architecture is shown in Fig. 3a. We also consider a similar architecture, but where the three-dimensional convolutions are replaced with Bottleneck blocks [37, 38] shown in Fig. 3b. We found that architectures using Bottleneck blocks result in lower LERs while using fewer parameters compared to the fully convolutional architectures shown in Fig. 4a. However despite the smaller parameter count, they come at a slower runtime cost compared to similar fully convolutional networks.

The relevant hyperparameters of the architecture in Fig. 3a are the number of convolutional layers, the kernel sizes, and the number of filters in each layer. For the architecture in Fig. 3b, we also have the p_{int} parameter used in the projection layers as well as p_{out} (note that we fix the scaling factor to $1/\sqrt{2L}$ with $L = 5$ used in the skip connections). These hyperparameters determine tradeoffs between pre-decoder latency and decoding accuracy, which are explored in Section IV. We denote the input data to the neural network by `trainX` and the labeled output data used during training by `trainY`. In this subsection, we describe the construction of `trainX`.

We first map stabilizer-measurement outcomes to a rectangular grid so that the syndrome data can be processed by convolutional layers. The mapping is performed in two steps. First, weight-6 and weight-4 stabilizers are embedded into a rectangular-grid representation, as illustrated in Fig. 4a. Second, each stabilizer measurement outcome is assigned to a data-qubit position on this grid, as shown in Fig. 4b. For all stabilizers except the red weight-4 stabilizers, the measurement outcome is mapped to the data qubit in the top-right position of its support, for both X- and Z-type stabilizers. For red weight-4 stabilizers, the measurement outcome is instead mapped to the data qubit in the top-left position of its support. The data qubits circled in red in Fig. 4b indicate the grid locations that receive stabilizer-measurement outcomes. If the corresponding outcome is -1, a value of 1 is inserted at that grid location; otherwise, the inserted value is 0. All other data-qubit locations are assigned the default value 0.

Let $\texttt{x\_type}(k, j)$ and $\texttt{z\_type}(k, j)$ denote the X- and Z-type detection-event grids, respectively, for the k-th syndrome-measurement round and the j-th shot. Detection events in round k are obtained by summing the mapped stabilizer values from rounds $k-1$ and k modulo 2. For a $D_x \times D_y$ rectangular grid, the first two channels of `trainX` are defined as

$$\texttt{trainX}(j, 1{:}D_x, 1{:}D_y, k, 1) = \texttt{x\_type}(k, j), \quad (1)$$
$$\texttt{trainX}(j, 1{:}D_x, 1{:}D_y, k, 2) = \texttt{z\_type}(k, j). \quad (2)$$

Similarly to Refs. [24, 31], we also encode geometric information about the lattice in `trainX`. These additional channels allow the network to distinguish between bulk and boundary syndrome statistics. We denote these channels by `x_present` and `z_present` for the X- and Z-type stabilizers, respectively. These two channels are identical except in the first and last syndrome-measurement rounds, where they depend on the basis in which the data qubits are prepared and measured, as explained below.

The construction of `x_present` and `z_present` uses the same grid mapping as in Fig. 4b. However, instead of mapping a stabilizer-measurement outcome to the corresponding grid location, we map the normalized stabilizer weight. The normalized weight is obtained by dividing the stabilizer weight by the maximum stabilizer weight, which is six in this case, so that all values lie in the interval $[0, 1]$. Grid locations that do not receive a stabilizer mapping, i.e., data qubits not circled in red in Fig. 4b, are assigned value 0. For the example shown in Fig. 4b,

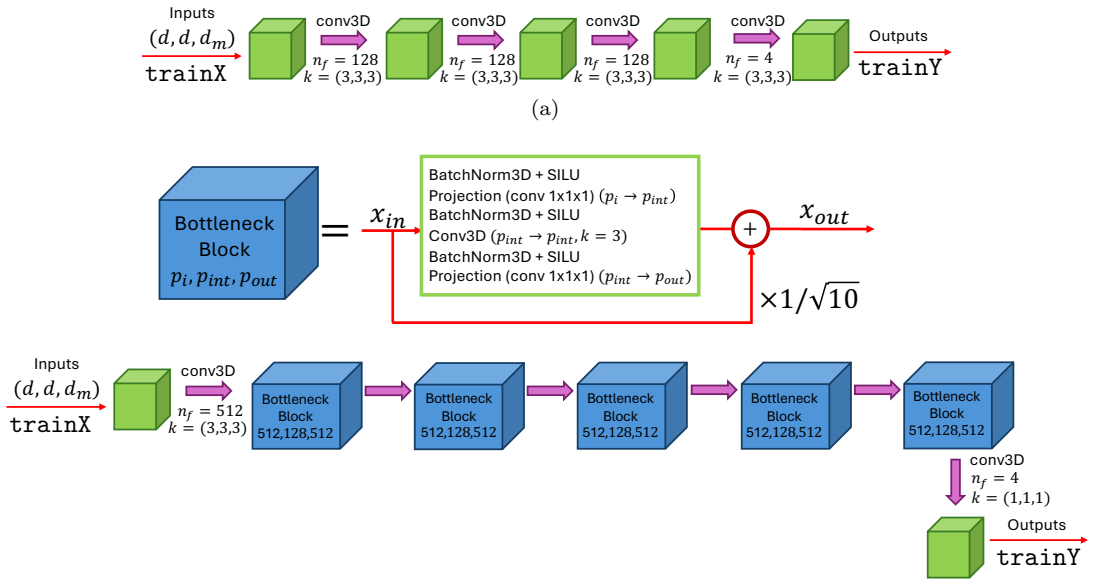


FIG. 3. (a) Illustration of the fully convolutional neural network architecture used to train our pre-decoders. The input data is labeled `trainX` and the output data labeled `trainY`. We use three-dimensional convolutional layers which can have varying kernel sizes and number of filters. The output of the last convolution layer goes through a sigmoid activation function in order to generate the spacelike and timelike predictions. Figure taken from Ref. [31]. (b) Alternative architecture which uses Bottleneck blocks taken from Refs. [37, 38]. We find that such architectures achieves lower LERs while using less parameters compared to the fully three-dimensional convolutional architecture with the same number of layers, conv3D kernel sizes and filters.

we have

$$\texttt{x\_present}(k) = \begin{bmatrix} 0 & 0 & 0 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2/3 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2/3 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 2/3 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 2/3 & 0 \\ 0 & 2/3 & 0 & 2/3 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

for $1 < k < d_m$, where d_m is the total number of syndrome-measurement rounds. In this range, $\texttt{z\_present}(k)$ is identical to $\texttt{x\_present}(k)$.

To encode temporal boundary information associated with state preparation and measurement, we modify the geometric channels in the first and last rounds. If the data qubits are prepared in $|0\rangle$ and measured in the Z-basis in round d_m, we set $\texttt{x\_present}(1)$ and $\texttt{x\_present}(d_m)$ to all-zero matrices. Similarly, if the data qubits are prepared in $|+\rangle$ and measured in the X-basis in round d_m, we set $\texttt{z\_present}(1)$ and $\texttt{z\_present}(d_m)$ to all-zero matrices. The geometric channels are then encoded as

$$\texttt{trainX}(j, 1{:}D_x, 1{:}D_y, k, 3) = \texttt{x\_present}(k), \tag{4}$$
$$\texttt{trainX}(j, 1{:}D_x, 1{:}D_y, k, 4) = \texttt{z\_present}(k). \tag{5}$$

B. Output training data and data processing

We follow Ref. [31] for generating the labeled output data `trainY`. As a first step, let $E^{(j)}(X)_{(i,k)} \in \{I, X\}$ for the i'th *data* qubit in round k and the j'th shot. We define the error difference in round k as $\tilde{X}^{(j)}_{i,k} = E^{(j)}(X)_{i,k} \oplus E^{(j)}(X)_{i,k-1}$. We then write

$$\tilde{X}^{(j)}_k \equiv (\tilde{X}^{(j)}_{(1,k)}, \cdots \tilde{X}^{(j)}_{(D_x D_y, k)}). \tag{6}$$

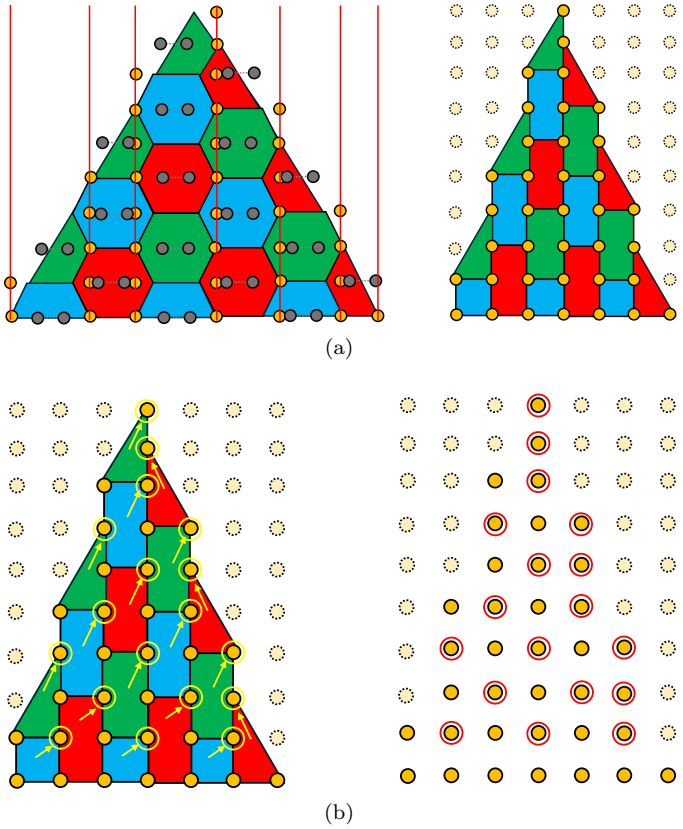
A similar definition for $\tilde{Z}^{(j)}_k$ is obtained for Z errors, i.e.

$$\tilde{Z}^{(j)}_k \equiv (\tilde{Z}^{(j)}_{(1,k)}, \cdots \tilde{Z}^{(j)}_{(D_x D_y, k)}). \tag{7}$$

The first two channels of `trainY` are then given by

$$\texttt{trainY}(j, 1{:}D_x, 1{:}D_y, k, 1) = \tilde{Z}^{(j)}_k, \tag{8}$$
$$\texttt{trainY}(j, 1{:}D_x, 1{:}D_y, k, 2) = \tilde{X}^{(j)}_k, \tag{9}$$

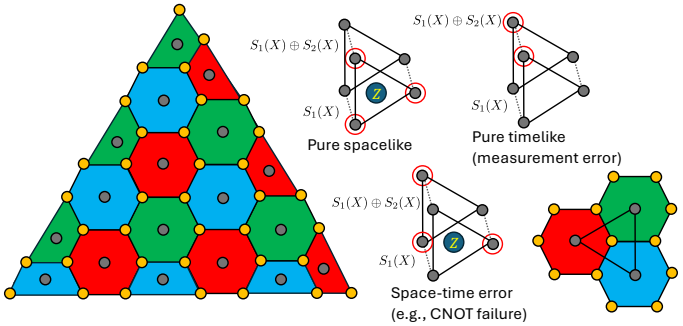
for the j-th shot and k-th training example. That is, the first two channels track changes in Z and X-type Pauli errors obtained by generating faults at each location of the syndrome extraction circuits for the color code and propagating the errors to obtain final data qubit errors of the $D_x \times D_y$ grid shown in Fig. 4. We note that special care is needed to track changes $\tilde{X}^{(j)}_k$ given the feedforward operations in the syndrome extraction circuits of Fig. 1. More details are given in Section III C.



(a)

(b)

FIG. 4. (a) Illustration of the mapping of the triangular color code to a two-dimensional grid. Vertical red lines are added to each data qubit at the bottom of the triangle. For each weight-4 and weight-6 stabilizer in between the red lines, data qubits closest to the right-most vertical line are mapped to that line. Similarly, data qubits closest to the left-most vertical line are mapped to that line. Such a mapping results in the figure on the right, resulting in rectangular weight-6 stabilizers and square or triangular weight-4 stabilizers. (b) Illustration of the mapping of stabilizer measurements to data qubit positions. A data qubit circled in red indicates that a stabilizer measurement is mapped to that data qubit.



(a)

FIG. 5. Example illustration of $S_1(X) \oplus S_2(X)$ computations used in Algorithm 1 for various fault mechanisms. Only pure timelike failures (which arise from measurement errors) and space-time failures (for instance CNOT gate failures) result in a non-zero value of $S_1(X) \oplus S_2(X)$. The examples provided are for failures arising in the bulk of the color code lattice. The illustration uses a single ancilla for simplicity to focus on just X-type stabilizers, but the methods apply in the same way to the circuit in Fig. 1.

Next we consider the labels for timelike failures which are encoded in channels three and four of `trainY`. To extract timelike failures from arbitrary fault mechanisms, we use Algorithm 1.

An illustration of the computation of $s_1(X) \oplus s_2(X)$ adapted to the color code lattice is shown in Fig. 5.

As was shown in Ref. [31], the time steps at which spacelike and space-time errors occur is of particular importance when constructing the labels for `trainY`. For instance, suppose a X error occurs in syndrome measure-

Algorithm 1 Timelike output channel generation

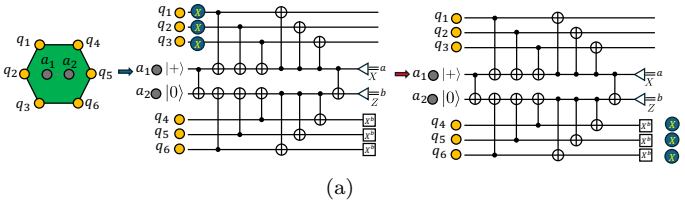
for $k = 1$ to $d_m - 1$ **do**
Let E_k be the errors generated by the noise model at each fault location in syndrome measurement round k.
Propagate E_k and compute:
X and Z stabilizer syndromes $s_1(X)$, $s_1(Z)$
Let $E^{(k)}_{\text{out}}$ be the output errors from propagating E_k.
Propagate $E^{(k)}_{\text{out}}$ and compute:
X and Z stabilizer syndromes $s_2(X)$, $s_2(Z)$
$\texttt{trainY}(j, 1{:}D, 1{:}D, k, 3) \leftarrow s_1(X) \oplus s_2(X)$
$\texttt{trainY}(j, 1{:}D, 1{:}D, k, 4) \leftarrow s_1(Z) \oplus s_2(Z)$

ment round $1 < k < d_m$ during the implementation of the circuit in Fig. 1b in the bulk of the color code lattice. Such an error anti-commutes with three Z-type stabilizers. However if the error occurs at a time step where it only gets detected in round $k+1$, `trainY` would contain the X error label in round k with the associated syndrome given in `trainX` in round $k+1$. Such scenarios can result in our networks learning patterns that result in residual timelike failures when applying corrections during inference time. Such a problem can be avoided by treating the X error as an input error in round $k+1$. More generally, we can use Algorithm 2 as a data processing protocol when generating `trainX` and `trainY`.

Algorithm 2 Data generation protocol

Let $e^{(0)} = \mathbf{0}$ be the empty error vector.
for $k = 1$ to $d_m - 1$ **do**
Initialize $e^{(k)} = \mathbf{0}$.
Let E_k be the full set of faults generated by the noise model at each fault location in syndrome measurement round k.
Append $e^{(k-1)}$ to E_k.
Let N_{E_k} be the number of faults in E_k, and $e^{(k)}_j$ be the j-th fault in E_k (with $1 \leq j \leq N_{E_k}$).
for $j = 1$ to N_{E_k} **do**
Propagate $e^{(k)}_j$ through the stabilizer measurement circuits of the color code.
Let $s_{e^{(k)}_j}$ be the measured syndrome arising from the fault $e^{(k)}_j$.
Let $|s_{e^{(k)}_j}|$ be the Hamming weight of $s_{e^{(k)}_j}$.
if $|s_{e^{(k)}_j}| > 0$ **then**
Update `trainX` and `trainY` as described in Sections III A and III B.
else
if $e^{(k)}_j$ results in a non-trivial data qubit error $e^{(k)}_{d_j}$ **then**
Append $e^{(k)}_{d_j}$ in time step 1 (so as to be treated as an input error) to $e^{(k)}$ and ignore updates to `trainY`.

Additional care is required when analyzing faults containing Y errors. For instance, the X component of a single Y error on a data qubit could be detected in round k, while the Z component is detected in round $k+1$. For this reason, prior to applying Algorithm 2, we decompose the relevant Y-containing faults into separately propagated Pauli components. A single-qubit Y error is represented as $X \oplus Z$, with the two components propagated independently. For two-qubit faults containing at least one Y error, the decomposition must respect how the X and Z components propagate through the CNOT gates in Fig. 1. By propagating two-qubit Paulis at each CNOT location, we obtain the rules shown in Table I. In particular, a rule of the form $P'_1P'_2 \to P_1P_2 \oplus P_3P_4$ means that, in Algorithm 2, the original fault $P'_1P'_2$ is replaced by two separately propagated components, P_1P_2 and P_3P_4. The resulting updates to `trainX` and `trainY` are then combined modulo two.



(a)

FIG. 6. Example of an artificial error difference stored in `trainY` due to the feedforward operations used in the stabilizer measurement circuits. In this example, an input weight-three X error with support on qubits q_1, q_2 and q_3 is mapped to the homologically equivalent weight-three error with support on qubits q_4, q_5 and q_6.

C. Error propagation step to remove the presence of artificial error differences

Recall that the first two channels of `trainY` store error differences as described in Eqs. (8) and (9). Due to the feedforward operations in the circuits shown in Fig. 1, artificial X error differences can be obtained if the propagation of X errors are not treated with care. An example is shown in Fig. 6 where an input weight-three X type error is mapped to a homologically equivalent output weight-three error with support on different qubits. In this case, an X error difference would be recorded in Eq. (9) even though no new errors occurred, thus lowering the performance of our pre-decoders. Note that input errors being mapped to equivalent output errors with support on different qubits only occurs for X-type errors (Z-type errors are immune to this issue since the feedforward operation only affects X-type errors).

Data qubit on control of CNOT, ancilla on target	Data qubit on target of CNOT, ancilla on control
$YZ \to ZZ \oplus XI$	$ZY \to ZZ \oplus IX$
YX no decomposition necessary	$XY \to XX \oplus IZ$
$YY \to ZZ \oplus XX$	YY no decomposition necessary

TABLE I. Complete Y error decomposition rules on two qubits following a CNOT gate. We use the format P_1P_2 where the Pauli P_1 is on the control of a CNOT and P_2 is on the target. The symbol $\oplus$ denotes a bookkeeping decomposition into separately propagated Pauli components whose contributions to the binary output labels are combined modulo two; it does not denote a physical tensor product or a probabilistic mixture.

Algorithm 3 Error propagation protocol

Initialize $\tilde{e}^0_{\text{out}} = \mathbf{0}$.
for $k = 1$ to $d_m - 1$ **do**
Let E_k be the full set of faults generated by the noise model at each fault location in syndrome measurement round k, with potential appended errors that were undetected in the previous round as described in Algorithm 2.
Let $\tilde{e}^k_{\text{temp}}$ be a binary vector for the output errors obtained by propagating E_k.
Compute $\tilde{e}^k_{\text{out}} = \tilde{e}^k_{\text{temp}} \oplus \tilde{e}^{k-1}_{\text{out}}$.
Compute X and Z stabilizer syndromes using $\tilde{e}^k_{\text{out}}$.
Error differences are computed in Eqs. (6) and (7) by simply taking the X and Z components of $\tilde{e}^k_{\text{temp}}$.

In Algorithm 3, we define an error-propagation protocol that explicitly tracks how faults evolve across rounds in order to prevent artificial differences in the resulting error configurations. This procedure is used in conjunction with Algorithm 2. The key idea is to avoid reintroducing errors from previous rounds as fresh input faults in the current round, which could otherwise be mapped to a different—though homologically equivalent—output error. Instead, feedforward corrections are computed using only the faults generated within the current round. Without this restriction, the weight-three input error in Fig. 6 would incorrectly induce a feedforward correction that transforms the output into a weight-six error.

D. Homological equivalence protocol for the color code

When constructing the output labeled data in `trainY`, several errors (both spacelike and timelike) have equivalent representations. In particular, let $g \in \mathcal{S}$ where $\mathcal{S}$ is the stabilizer group of the color code. We say than an error E' is homologically equivalent in the spacelike sense to an error E if we can write $E' = gE$. As was shown in Refs. [24, 29, 31], choosing a fixed representation of homologically equivalent errors can significantly improve the training performance of our neural networks. Below we present a spacelike homological equivalence protocol for the color code which leads to greatly improved performance of our AI-based pre-decoders. We note that in Ref. [31] a timelike homological equivalence protocol was also developed. However we found numerically that combining a spacelike and timelike homological equivalence protocol for the color code led to inferior results compared to simply using the spacelike protocol presented below.

1. *Spacelike homological equivalence protocol*

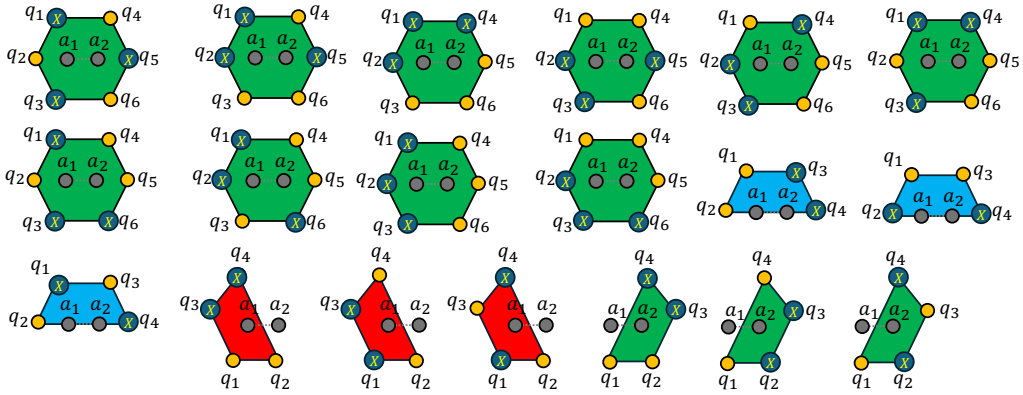
When writing $E' = gE$, two things can happen. The hamming weight $|E'|$ can be reduced relative to E, or we can have $|E'| = |E|$ with errors shuffled along the support of the stabilizer g. If g is an X (Z) type stabilizer, applying g to E when $|E'| < |E|$ is referred to as `weightReductionX` (`weightReductionZ`), and when $|E'| = |E|$ as `fixEquivalenceX` (`fixEquivalenceZ`).

Let $g_6^{(X)}(j)$ correspond to the j-th weight-6 X-type stabilizer for the color code. Similarly, let $g_4^{(X)}(j)$ correspond to the j-th weight-4 X-type stabilizer for the color code. We define `weightReductionX` as follows. Let $d_6^{(X)}$ ($d_4^{(X)}$) be the number of weight-6 (weight-4) X-type stabilizers for a distance d color code. For all $1 \leq j_1 \leq d_6^{(X)}$, we apply $g_6^{(X)}(j_1)$ to `trainY`. For all $1 \leq j_2 \leq d_4^{(X)}$, we apply $g_4^{(X)}(j_2)$ to `trainY`. If the number of 1's in `trainY` is reduced, we accept the change; otherwise, we leave `trainY` unchanged.

Next we describe the construction of `fixEquivalenceX`. For each weight-6 X-type stabilizer, we verify if a weight-3 X error is in its support. If yes, we determine which configuration in Fig. 7 the error belongs to. If the configuration is not as in the ones shown, we modify the error in `trainY` to be the equivalent representation of one of the ten configurations shown. We perform an identical operation but for weight-2 X-type errors in the support of weight-4 X stabilizers using the configurations shown in Fig. 7.

Next define the function `simplifyX` which applies `weightReductionX` and `fixEquivalenceX` (in this order) to all X-type stabilizers of the color code. The function `simplifyX` is repeatedly applied until a steady state for `trainY` is achieved. By steady state, we mean that applying `simplifyX` to `trainY` no longer produces a change to `trainY`.

We note that equivalent functions `weightReductionZ`, `fixEquivalenceZ` and `simplifyZ` can be used for Z errors by simply replacing all X operators with Z operators given the properties of stabilizers for the color code. As such, the full spacelike homological equivalence func-



(a)

FIG. 7. Configurations used in `fixEquivalenceX` for X errors in the support of weight-6 and weight-4 X-type stabilizers. We pair each weight-3 X error with its homologically equivalent error obtained by multiplying by the corresponding weight-6 stabilizer. The same rules are used for Z-type stabilizers.

tion is executed by repeatedly applying `simplifyX` and `simplifyZ` until a steady state for `trainY` is achieved.

Lastly, we remark that not all choices of `fixEquivalenceX` results in convergence when repeatedly applying `simplifyX`. That is, repeatedly applying `simplifyX` could result in errors continuously being shifted around without ever reaching a steady state. However, the choice of `fixEquivalenceX` in Fig. 7 does lead to convergence when repeatedly applying `simplifyX`.

IV. NUMERICAL RESULTS

In this section we present numerical results for the family of pre-decoder models trained using the fully convolutional neural networks summarized in Table II and illustrated in Fig. 3a as well as the model using five layers of Bottleneck blocks shown in Fig. 3b. We refer to the model with Bottleneck layers as model B. Note that model B has 2,936,580 parameters which is considerably less than model 5 despite having the same receptive field and $p_i = p_{out} = 512$.

The pre-decoder models trained using the fully three-dimensional convolutional architecture use the hyperparameters listed in Table III with data generated on the fly using a code distance d matching each model's receptive field. Model B uses the same hyperparameters as in Table III except that the learning rate is chosen to be 10^{-5} and the activation functions are SiLU instead of GeLU (see Fig. 3b). Unless otherwise stated, simulations throughout this section employ the following depolarizing circuit-level noise model:

- A $|0\rangle$ ($|+\rangle$) state preparation is followed by an X (Z) error with probability $2p/3$.
- Prior to each Z (X) basis measurement, an X (Z) error occurs with probability $2p/3$.
- With probability p, each two-qubit gate is followed by a two-qubit Pauli error drawn uniformly from $\{I, X, Y, Z\}^{\otimes 2} \setminus \{I \otimes I\}$.
- During idle locations associated with either CNOT gates or state-preparation and measurement, a Pauli error is drawn uniformly from $\{X, Y, Z\}$ with probability p.

A. Pre-decoder logical error rates and syndrome densities with Chromobius as the global decoder

Plots of the per-round logical error rates (LERs) for Chromobius and the pre-decoder + Chromobius pipeline are shown in Fig. 8 when using models 1 and 5 (results for model B are given in Fig. 12). We illustrate results for model 1, which has the fastest inference time on a GPU, and model 5, which provides the best LER performance.

Several observations are immediately apparent. For both models, the pre-decoder + Chromobius pipeline achieves lower LERs than Chromobius alone across all sampled physical error rates, including $p = 0.1\%$, which is relevant for state-of-the-art hardware architectures. Most importantly, in nearly all cases, the relative LER improvement increases with code distance, even though the pre-decoder models were trained only at a fixed distance equal to the receptive-field size. These trends are quantified more explicitly in Table IV. Note that in Table IV, we also provide the LER improvement at $p = 0.3\%$ for model B which gives the best LER improvements across all considered models (although at the cost of slower runtimes as shown in Section IV B).

The improvements are substantial. For example, at $p = 0.3\%$, the pre-decoder + Chromobius pipeline at $d = 13$ achieves a lower LER than raw Chromobius decoding at $d = 31$. This corresponds to obtaining superior logical performance using only 253 physical qubits (127 data qubits and 126 ancillas) instead of 1441 physical

	num_filters	kernel_size	RF size	num_params
Model 1	[128,128,128,4]	[3,3,3,3]	9	912,272
Model 2	[256,256,256,4]	[3,3,3,3]	9	3,595,012
Model 3	[128,128,128,4]	[5,5,5,5]	17	4,224,388
Model 4	[128,128,128,128,128,4]	[3,3,3,3,3,3]	13	1,797,764
Model 5	[256,256,256,256,256,4]	[3,3,3,3,3,3]	13	7,134,468

TABLE II. Pre-decoder models considered in this work. The size of the vectors used for `num_filters` and `kernel_size` indicate how many 3DConv layers are used. The entries in `num_filters` and `kernel_size` indicate the number of filters and kernel size used in that given layer. Note that if an entry in the j-th column of `kernel_size` is K, a kernel size of $K \times K \times K$ is used in that layer. All models use stride 1 and no dilation.

Hyperparameters	**Values**
Shots per epoch	16,777,216
Number of epochs	400
Batch size per GPU	Epoch 1: 256, Epoch $\geq$ 2: 1024
Number of GPUs	4
Optimizer	Lion: Weight decay $= 10^{-7}$, beta2 $= 0.95$
Learning rate schedule	Warmup then decay (100 warmup steps). Apply $\gamma = 0.7$ at milestones $[0.25, 0.5, 1.0]$
Learning rates	Model 1 $= 2 \times 10^{-5}$, Model 2 $= 1 \times 10^{-5}$, Model 4 $= 1 \times 10^{-5}$, Model 5 $= 1 \times 10^{-5}$
Activation function	GeLU (tanh approximation)
Dropout	0.01
Exponential moving average (ema)	decay $= 0.0001$
Physical error rate	Model 1 $= 0.003$, Others $= 0.004$

TABLE III. Hyperparameters used to train models 1 to 5 from Table II. The $\gamma = 0.7$ is applied to the learning rate at milestones $[0.25, 0.5, 1.0]$. For instance, the first milestone 0.25 indicates that at 25% of training steps, the learning rate becomes $0.7 \times$ base. The tanh approximation of GeLU uses the function $\mathrm{GeLU}(x) \approx 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$.

Model	$d = 5$	$d = 9$	$d = 13$	$d = 17$	$d = 21$	$d = 31$
Model 1	1.33x	1.51x	2.13x	3.18x	5.03x	18.58x
Model 4	1.68x	2.23x	3.87x	7.67x	15.73x	124.71x
Model 5	1.82x	2.64x	4.84x	10.61x	23.78x	223.72x
Model B	1.79x	2.88x	5.56x	13.16x	31.21x	**347.74**x

TABLE IV. LER improvement factor (X-basis) for models 1, 4 and 5 of Table II followed by Chromobius compared to Chromobius alone. All data is obtained at $p = 0.3\%$.

qubits (721 data qubits and 720 ancillas), while simultaneously achieving significantly lower decoding runtimes (see Section IV B). We additionally observe an increase in the effective threshold when using pre-decoding prior to Chromobius.

This improvement can also be understood in terms of the effective distance scaling. For standalone Chromobius, which implements a Moebius decoder, the logical failure probability is expected to scale approximately as $p_L(d, p) \sim p^{\alpha d}$ with $\alpha \simeq 3/7$ [39]. Fitting the raw Chromobius total logical failure rates in our data gives $\alpha \approx 0.43$, in close agreement with this expectation. Repeating the same fit for the pre-decoder + Chromobius pipeline yields a larger effective exponent. For model 5, fitting through $p \leq 0.3\%$ gives $\alpha \approx 0.49$ for both X- and Z-basis measurements, matching the scaling of [40] and corresponding to an effective scaling closer to $p^{3.4d/7}$. This indicates that the pre-decoder improves not only the prefactor but also the observed distance scaling of the combined decoder. However, this scaling still does not saturate the optimal expectation $p_L(d, p) \sim p^{(d+1)/2}$. Thus, the pre-decoder moves the practical scaling closer to the optimal behavior, but does not fully reach it over the range of distances and physical error rates studied here.

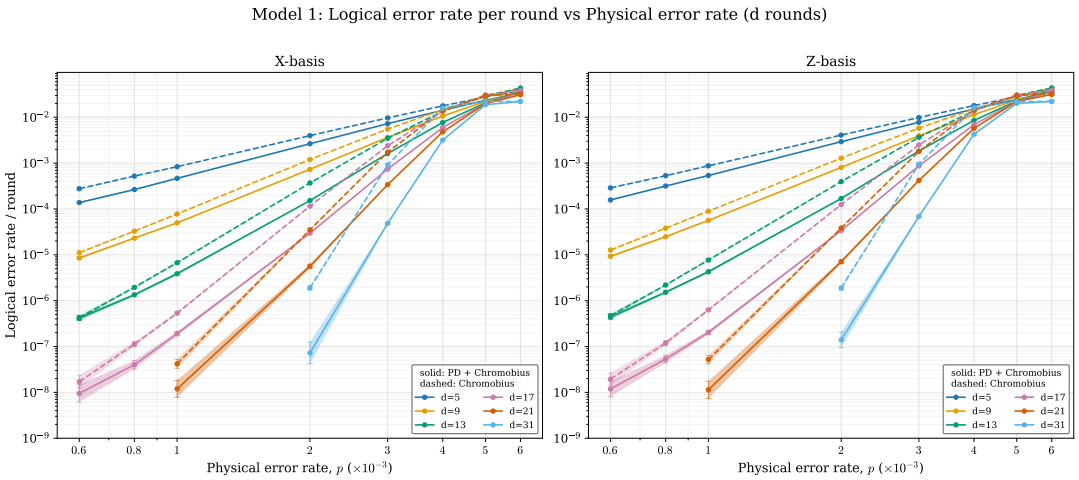
A distance-d triangular color code using the syndrome-extraction circuits of Fig. 1 has a total qubit count (data + ancilla) given by

$$n_{\mathrm{color}} = \frac{3d^2 - 1}{2}. \tag{10}$$

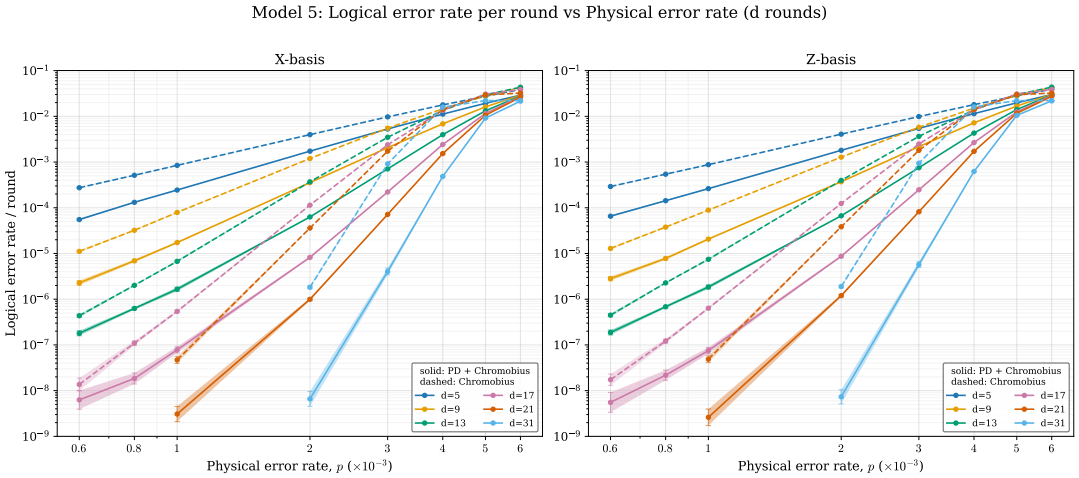
For comparison, the rotated surface code [3] requires

$$n_{\mathrm{surface}} = 2d^2 - 1. \tag{11}$$

In Fig. 10, we plot the number of physical qubits, computed using Eqs. (10) and (11), required to reach a specified target logical error rate for both surface codes and color codes. We show results for several decoders. The figure demonstrates that, for color-code memories, the model B + Chromobius decoder substantially reduces the required number of physical qubits relative to Chromobius alone. However, for pure quantum-memory benchmarks, Fig. 10 also shows that surface codes can reach a given



(a)



(b)

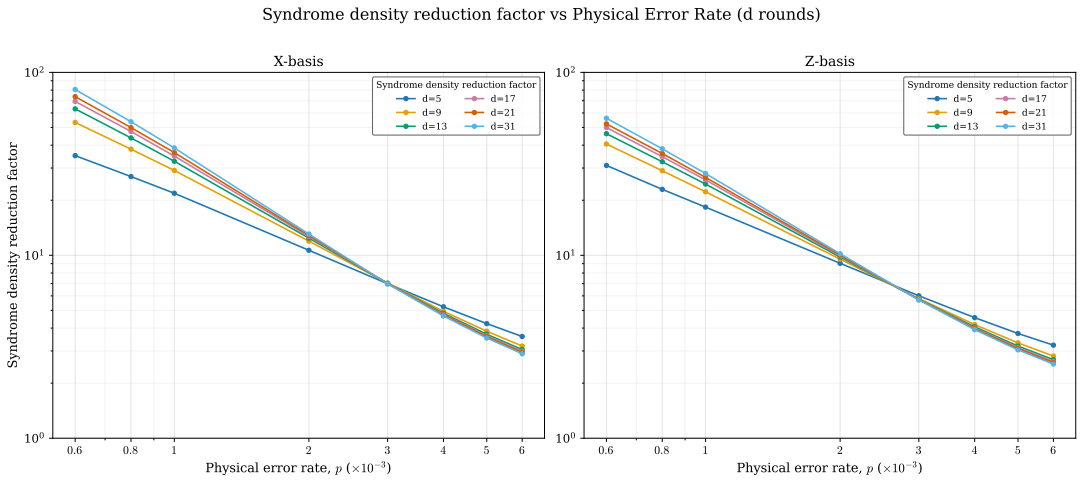
FIG. 8. Plots of per-round LER for Chromobius (dashed lines) vs per-round LER of a pre-decoder model followed by Chromobius (solid lines). Per-round LERs were computed by calculating the LER for d rounds and dividing the result by d.

Model	**d=13,** $p = 0.001$ **(μs/round)**	**d=13,** $p = 0.003$ **(μs/round)**	**d=21,** $p = 0.001$ **(μs/round)**	**d=21,** $p = 0.003$ **(μs/round)**	**d=31,** $p = 0.001$ **(μs/round)**	**d=31,** $p = 0.003$ **(μs/round)**
Chromobius	9.998	36.694	41.455	161.203	144.794	577.129
Chromobius after model 1 (GeLU)	1.739	11.527	6.965	44.600	23.360	156.279
Chromobius after model 4 (GeLU)	1.242	7.484	5.466	26.314	18.530	87.312
Chromobius after model 5 (GeLU)	1.050	6.321	4.450	22.424	15.718	72.987
Chromobius after model B	1.058	6.125	4.607	20.836	16.125	64.268
Pre-decoder model 1 (GeLU)	3.44	3.44	3.19	3.19	3.73	3.73
Pre-decoder model 4 (GeLU)	4.23	4.23	4.08	4.08	4.94	4.94
Pre-decoder model 5 (GeLU)	6.12	6.12	6.52	6.52	9.17	9.17
Pre-decoder model B (SiLU)	8.74	8.74	9.44	9.44	14.39	14.39

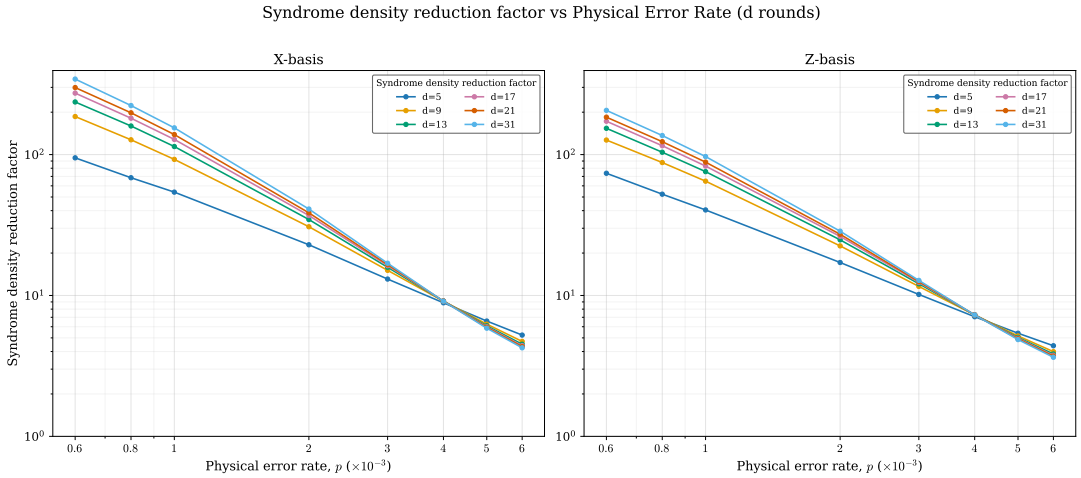
TABLE V. Comparison of runtimes for Chromobius (both with and without syndromes processed by pre-decoder models) and pre-decoder models. All results correspond to the task of decoding a single (batch size = 1) $d \times (d + (d-1)/2) \times d$ block, and we report averaged runtimes per syndrome measurement round. Chromobius runtimes are computed using a Grace Neoverse-V2 CPU. The label "Chromobius after model X" refers to Chromobius runtimes after processing syndromes by the pre-decoder model X (i.e. one of the 5 models in Table II). GPU runtimes for all five pre-decoder models are computed using an NVIDIA GB300 GPU using TensorRT with FP8 precision. All results are for X-basis measurements.

target logical error rate using fewer physical qubits than color codes, even without the use of a pre-decoder.

Nevertheless, logical operations implemented using transversal gates and lattice surgery can be substantially more efficient for color codes than for surface codes. Thus, these results suggest that AI-based pre-decoding may significantly improve the competitiveness of color codes for universal fault-tolerant quantum computation, even if surface codes retain an advantage in the pure-memory setting. The qubit overhead of color codes could po-



(a)



(b)

FIG. 9. Plots of the syndrome density reduction factor for models 1 and 5 as a function of the physical error rate p at various code distances. In (a) we show results for model 1 and in (b) for model 5.

tentially be reduced further by using the "middle-out" syndrome-extraction circuits of Ref. [7], which avoid dedicated ancilla qubits. Additional exploration of model architectures and training strategies may further narrow the remaining qubit-count gap between surface codes and color codes.

Finally, syndrome-density reduction (SDR) plots for models 1 and 5 are shown in Fig. 9. For $p \lesssim 0.1\%$, SDRs exceeding two orders of magnitude are achieved, meaning that more than 99% of syndromes have been canceled and resulting in substantial Chromobius runtime reductions (see Section IV B for exact runtimes). Similar to the LER results, SDR performance generally improves with increasing code distance.

d	X-basis	Z-basis
5	0.351	0.336
9	0.464	0.352
13	1.050	0.621
17	2.227	1.203
21	4.450	2.180
31	15.718	8.187

TABLE VI. Chromobius runtimes after applying Model 5 at $p = 0.1\%$. Runtimes are reported in μs per round and are separated by measurement basis.

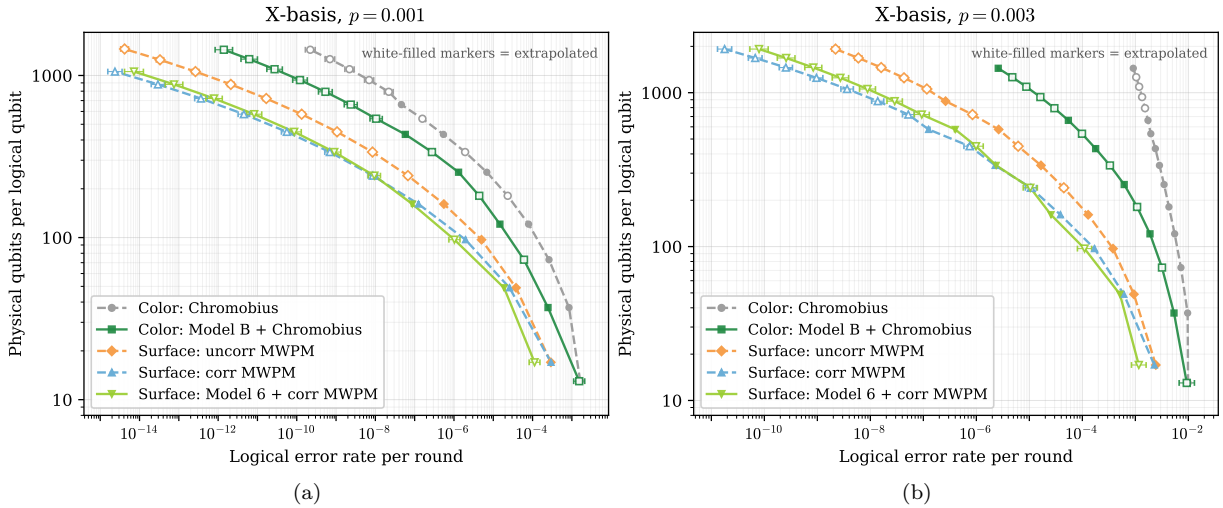


FIG. 10. Physical qubit count required to achieve a desired logical failure rate for both surface codes and color codes with various decoders. In (a), numbers are for physical error rates $p = 0.001$ and in (b) for $p = 0.003$.

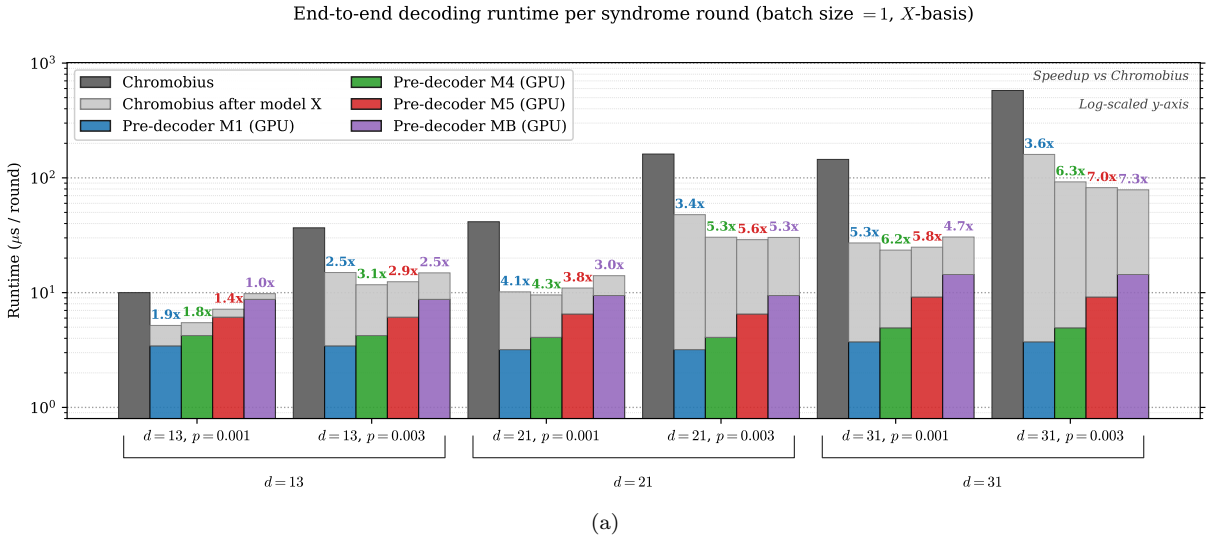


FIG. 11. Total speedup factors when using a pre-decoder (model MX with GeLU activation) + Chromobius compared to Chromobius alone. The speedup is defined as the ratio between raw Chromobius runtimes and the sum of pre-decoder inference runtimes plus Chromobius runtimes after pre-decoding, averaged over both bases (see Table V and Table VI). The largest speedup factor for each input setting is shown in bold.

B. Pre-decoder logical runtimes with Chromobius as the global decoder

In this section, we characterize the runtimes of Chromobius both with and without pre-decoding, as well as the runtimes of the pre-decoder models themselves across various code distances and physical error rates. We then quantify the overall runtime reduction achieved by the pre-decoder + Chromobius pipeline relative to standalone Chromobius decoding.

Table V summarizes the runtimes of Chromobius, the pre-decoder + Chromobius pipeline, and the pre-decoder models 1, 4, 5 and B for $d = 13, 21$ and 31 at physical error rates $p = 0.1\%$ and $p = 0.3\%$ for X-basis memory experiments. At $p = 0.1\%$, Model 5 yields the largest reductions in Chromobius runtimes, achieving up to a 9.5x improvement at $d = 13$. At $p = 0.3\%$, model B yields the largest reductions in Chromobius runtimes (a 9x improvement at $d = 31$). However, model B is also the slowest of the considered pre-decoders (even though it uses 2,936,580 parameters compared to 7,134,468 for model 5), while model 1 provides the lowest pre-decoder latency. We focus on $p = 0.1\%$ and $p = 0.3\%$ since these are relevant error rates for near-term hardware architectures.

We observe a consistent Chromobius runtime asymmetry that is basis-dependent and that gets larger with

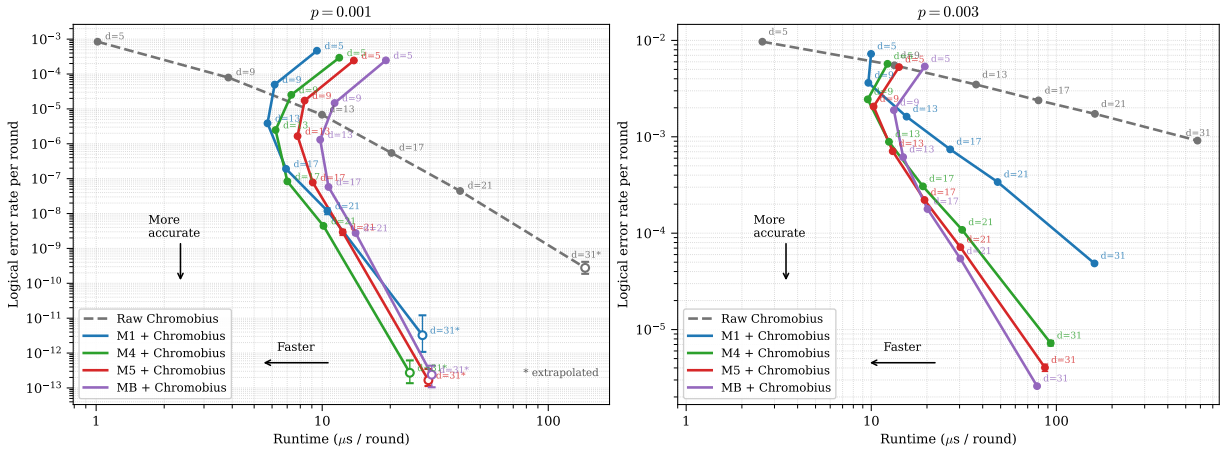


FIG. 12. End-to-end per-round LERs and single-shot (batch size 1) runtimes for Chromobius and pre-decoder models + Chromobius with representative physical error rates $p = 0.001$ (left) and $p = 0.003$ (right) for X-basis. Pre-decoder models M1, M4, M5 and MB run at FP8 precision and were timed on a single GB300 GPU, while Chromobius was timed on a single Grace Neoverse-V2 CPU.

decreasing physical error rates and increasing code distances. To exemplify it, Table VI shows averaged Chromobius runtimes after applying Model 5 at $p = 0.1\%$ for both X and Z bases. While the asymmetry is barely noticeable at $d = 5$, it grows to almost a 2x difference between X and Z-basis decoding runtimes at $d = 31$. As observed in Ref. [7], the Chromobius decoder when used with the superdense syndrome extraction circuits given in Fig. 1 results in asymmetric LERs between the X and Z basis. As such it should not be surprising to see these results amplified when using our pre-decoders given the improvement in LERs and runtimes.

In Fig. 11, we summarize the total end-to-end runtime improvements achieved by the pre-decoder + Chromobius pipeline for models 1, 4, 5 and B. The largest observed overall speedup is 7.33x, obtained using model B at $d = 31$ and $p = 0.3\%$. All four models exhibit parameter regimes in which they outperform the others. Importantly, the overall speedup generally increases with code distance. For fixed code distance, larger speedups are typically observed at higher physical error rates due to the increased syndrome density, which significantly slows Chromobius decoding. In contrast, pre-decoder runtimes are completely insensitive to the physical error rate.

In Fig. 12, we plot the end-to-end per-round LERs as a function of total decoding runtime for Chromobius and the pre-decoder + Chromobius pipelines for X-basis measurements. These plots illustrate the tradeoff between logical performance and decoding latency. For example, in Fig. 12a which shows data at $p = 0.1\%$, pre-decoding improves LERs for all considered distances, but for $d \leq 9$, the total runtime remains larger than that of standalone Chromobius decoding. However, for $d \geq 13$, all four considered pre-decoder models simultaneously improve both LER and runtime relative to Chromobius alone. At $p = 0.3\%$, the simultaneous runtime and LER improvement regime extends to $d \geq 9$ (except for model B which offers an advantage for $d \geq 13$), as shown in Fig. 12b.

The $d = 31$, $p = 0.1\%$ logical error rates shown in Fig. 12a are extrapolated. For standalone Chromobius, we use the scaling reported for the Moebius decoder, $p_L(d, p) \propto p^{3d/7}$ [39], applied to the measured total logical failure rates at $d = 31$ and $p = 0.2\%, 0.3\%$. For the pre-decoder curves, we use an analogous local-slope extrapolation. We assume that for d larger than the receptive field of the corresponding model the effective exponent per unit distance is approximately stable. We therefore estimate local power-law slopes from measured total LERs at post-receptive-field distances, rescale the inferred exponent density to $d = 31$, and extrapolate the measured $d = 31$ data down to $p = 0.1\%$. These extrapolated points should be interpreted as order-of-magnitude estimates rather than direct measurements.

V. CONCLUSION

In this work, we introduced an AI-based pre-decoder for triangular color codes. We developed a novel architecture for mapping color-code stabilizers onto a two-dimensional grid suitable for convolutional neural-network processing. We additionally introduced several data-processing techniques that substantially improve training performance, particularly for the complex syndrome-extraction circuits containing feedforward operations.

Using Chromobius as the global decoder, we demonstrated that the full pre-decoder + Chromobius pipeline achieves substantial reductions in both logical error rates (LERs) and decoding runtimes relative to standalone Chromobius decoding. Importantly, these gains generally be-

come more pronounced as the code distance increases across nearly all sampled physical error rates. For example, at $d = 31$ and $p = 0.3\%$, our pipeline achieves a 347x reduction in LER together with a 7.33x runtime improvement relative to Chromobius alone.

Several important directions for future work remain. First, broader model exploration may yield further improvements in both decoding performance and runtime. For example, incorporating projection layers between convolutional layers could reduce parameter counts while simultaneously improving model expressivity and performance [37, 38]. Additional GPU-level optimizations, including the use of lower-precision arithmetic such as FP4, may further reduce pre-decoder inference latency, particularly on next-generation hardware architectures designed for low-precision computation. Further, current work is being done to develop a parallel implementation of pre-decoders on GPUs. Early results show substantial improvements in both throughput and latency compared to decoding entire volumes resulting in runtimes well below 1μs.

Finally, extending AI-based pre-decoders to fully support lattice-surgery operations within parallel block-wise decoding frameworks in both space and time will be crucial for scalable real-time fault-tolerant quantum computation. We believe that the results presented in this work provide further evidence that AI-assisted decoding may significantly improve the practicality and competitiveness of color-code architectures for universal fault-tolerant quantum computing.

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, Journal of Mathematical Physics **43**, 4452 (2002).

[2] 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).

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

[4] H. Bombin and M. A. Martin-Delgado, Topological quantum distillation, Phys. Rev. Lett. **97**, 180501 (2006).

[5] A. Kubica and M. E. Beverland, Universal transversal gates with color codes: A simplified approach, Phys. Rev. A **91**, 032330 (2015).

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

[7] C. Gidney and C. Jones, New circuits and an open source decoder for the color code, arXiv e-prints , arXiv:2312.08813 (2023), arXiv:2312.08813 [quant-ph].

[8] A. G. Fowler and C. Gidney, Low overhead quantum computation using lattice surgery, arXiv e-prints , arXiv:1808.06709 (2018), arXiv:1808.06709 [quant-ph].

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

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

[11] D. Litinski, A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery, Quantum **3**, 128 (2019), 1808.02892.

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

[13] C. Chamberland and E. T. Campbell, Circuit-level protocol and analysis for twist-based lattice surgery, Phys. Rev. Research **4**, 023090 (2022).

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

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

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

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

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

[19] C. Gidney, N. Shutty, and C. Jones, Magic state cultivation: growing T states as cheap as CNOT gates, arXiv e-prints , arXiv:2409.17595 (2024), arXiv:2409.17595 [quant-ph].

[20] J. Edmonds, Paths, trees, and flowers, Canadian Journal of Mathematics **17**, 449–467 (1965).

[21] O. Higgott, Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching, ACM Transactions on Quantum Computing **3**, 10.1145/3505637 (2022).

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

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

[24] 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 Science and Technology **8**, 045011 (2023).

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

[26] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen, Scalable Surface-Code Decoders with Parallelization in Time, PRX Quantum **4**, 040344 (2023), arXiv:2209.09219 [quant-ph].

[27] S. Koutsioumpas, T. Noszko, H. Sayginel, M. Webster, and J. Roffe, Colour Codes Reach Surface Code Performance using Vibe Decoding, arXiv e-prints , arXiv:2508.15743 (2025), arXiv:2508.15743 [quant-ph].

[28] A. W. Senior, T. Edlich, F. J. H. Heras, L. M. Zhang, O. Higgott, J. S. Spencer, T. Applebaum, S. Black-

well, J. Ledford, A. Žemgulytė, A. Žídek, N. Shutty, A. Cowie, Y. Li, G. Holland, P. Brooks, C. Beattie, M. Newman, A. Davies, C. Jones, S. Boixo, H. Neven, P. Kohli, and J. Bausch, A scalable and real-time neural decoder for topological quantum codes, arXiv e-prints , arXiv:2512.07737 (2025), arXiv:2512.07737 [quant-ph].

[29] 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).

[30] S. Gicev, L. C. L. Hollenberg, and M. Usman, Fully convolutional 3D neural network decoders for surface codes with syndrome circuit noise, arXiv e-prints , arXiv:2506.16113 (2025), arXiv:2506.16113 [quant-ph].

[31] C. Chamberland, J. Olle, M. Li, S. Thornton, and I. Baratta, Fast and accurate AI-based pre-decoders for surface codes, arXiv e-prints , arXiv:2604.12841 (2026), arXiv:2604.12841 [quant-ph].

[32] H. Bombín, Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes, New Journal of Physics **17**, 083002 (2015).

[33] P. Baireuther, M. D. Caio, B. Criger, C. W. J. Beenakker, and T. E. O'Brien, Neural network decoder for topological color codes with circuit level noise, New Journal of Physics **21**, 013003 (2019).

[34] R. Chao and B. W. Reichardt, Quantum error correction with only two extra qubits, Phys. Rev. Lett. **121**, 050502 (2018).

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

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

[37] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (2016) pp. 770–778.

[38] A. Gu, J. P. Bonilla Ataides, M. D. Lukin, and S. F. Yelin, Scalable Neural Decoders for Practical Fault-Tolerant Quantum Computation, arXiv e-prints , arXiv:2604.08358 (2026), arXiv:2604.08358 [quant-ph].

[39] K. Sahay and B. J. Brown, Decoder for the triangular color code by matching on a möbius strip, PRX Quantum **3**, 010310 (2022).

[40] S.-H. Lee, A. Li, and S. D. Bartlett, Color code decoder with improved scaling for correcting circuit-level noise, Quantum **9**, 1609 (2025).