Data-driven decoding of quantum error correcting codes using graph neural networks

Citation for the original published paper (version of record):
Lange, M., Havström, P., Srivastava, B. et al (2025). Data-driven decoding of quantum error correcting codes using graph neural networks. Physical Review Research, 7(2).
http://dx.doi.org/10.1103/PhysRevResearch.7.023181

Data-driven decoding of quantum error correcting codes using graph neural networks

Moritz Lange,[1,*] Pontus Havström,[1] Basudha Srivastava,[1,2] Isak Bengtsson,[3] Valdemar Bergentall,[1]
Karl Hammar,[1] Olivia Heuts,[1] Evert van Nieuwenburg,[4,†] and Mats Granath[1,‡]
[1]*Department of Physics, University of Gothenburg, Gothenburg, Sweden*
[2]*Quantinuum, Terrington House, 13-15 Hills Rd, Cambridge CB2 1NL, United Kingdom*
[3]*Department of Physics, Chalmers University of Technology, Gothenburg, Sweden*
[4]*Leiden Inst. of Advanced Computer Science, Leiden University, Leiden, Netherlands*

(Received 22 September 2023; accepted 1 May 2025; published 23 May 2025)

To leverage the full potential of quantum error-correcting stabilizer codes it is crucial to have an efficient and accurate decoder. Accurate, maximum likelihood, decoders are computationally very expensive whereas decoders based on more efficient algorithms give sub-optimal performance. In addition, the accuracy will depend on the quality of models and estimates of error rates for idling qubits, gates, measurements, and resets, and will typically assume symmetric error channels. In this work, we explore a model-free, data-driven, approach to decoding, using a graph neural network (GNN). The decoding problem is formulated as a graph classification task in which a set of stabilizer measurements is mapped to an annotated detector graph for which the neural network predicts the most likely logical error class. We show that the GNN-based decoder can outperform a matching decoder for circuit level noise on the surface code given only the simulated data, while the matching decoder is given full information of the underlying error model. Although training is computationally demanding, inference is fast and scales approximately linearly with the space-time volume of the code. We also find that we can use large, but more limited, datasets of real experimental data for the repetition code, giving decoding accuracies that are on par with minimum weight perfect matching. The results show that a purely data-driven approach to decoding may be a viable future option for practical quantum error correction, which is competitive in terms of speed, accuracy, and versatility.

DOI: 10.1103/PhysRevResearch.7.023181

I. INTRODUCTION

Quantum Error Correction (QEC) is foreseen to be a vital component in the development of practical quantum computing [1–5]. The need for QEC arises due to the susceptibility of quantum information to noise, which can rapidly accumulate and corrupt the final output. Unlike noise mitigation schemes where errors are reduced by classical post-processing [6–8], QEC methods encode quantum information in a way that allows for the detection and correction of errors without destroying the information itself. A prominent framework for this is topological stabilizer codes, such as the surface code, for which the logical failure rates can be systematically suppressed by increasing the size of the code if the intrinsic error rates are below some threshold value [9–13].

Stabilizer codes are based on a set of commutative, typically local, measurements that project an n-qubit state to a lower dimensional code space representing one or more logical qubits. Errors take the state out of the code space and are then indicated by a syndrome, corresponding to stabilizer violations. The syndrome needs to be interpreted in order to gauge whether a logical bit or phase flip may have been incurred on the logical qubit. Interpreting the syndrome, to predict the most likely logical error, requires both a decoder algorithm and, traditionally, a model of the qubit error channels. The fact that measurements may themselves be noisy, makes this interpretation additionally challenging [10,13].

Efforts are under way to realize stabilizer codes experimentally using various qubit architectures [14–30]. In [28], code distance 3 and 5 surface codes were implemented, using 17 and 49 superconducting qubits, respectively. After initialization of the qubits, repeated stabilizer measurements are performed over a given number of cycles capped by a final round of single qubit measurements. The results are then compared with the initial state to determine whether errors have caused a logical bit- (or phase-) error. The decoder analyses the collected sets of syndrome measurements in post-processing, where the fraction of correct predictions gives a measure of the logical accuracy. The better the decoder, the higher the coherence time of the logical qubit, and in [28] a computationally costly tensor network based decoder was used to maximize the logical fidelity of the distance 5 code compared to the distance 3 code. However, with the objective of moving from running and benchmarking a quantum memory to using it for universal quantum computation, it will be

[*] Contact author: moritz.lange@physics.gu.se
[†] Contact author: evert.vn@lorentz.leidenuniv.nl
[‡] Contact author: mats.granath@physics.gu.se

necessary to do error correction both with high accuracy and in real time.

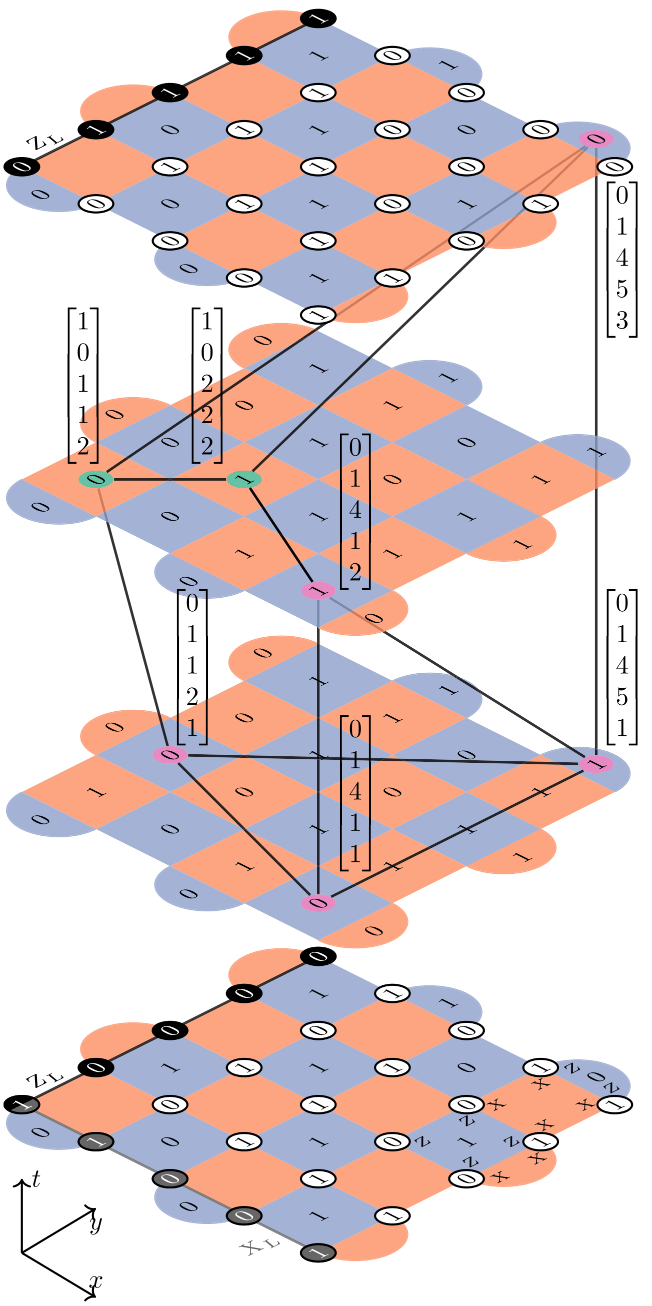
In the present work, we explore the viability of using a purely data-driven approach to decoding, based on the potential of generating large amounts of experimental data. We use a graph neural network (GNN) which is well suited for addressing this type of data. Namely, a single data point, as in [28], consists of a set of "detectors", i.e., changes in stabilizer measurements from one cycle to the next, together with a label indicating the measured logical bit- or phase-flip error. This can be represented as a labeled graph with nodes that are annotated by the information on the type of stabilizer and the space-time position of the detector, as shown in Fig. 1. The maximum degree of the graph can be capped based on removing edges between distant detectors, keeping only a fixed maximum number of neighboring nodes. The latter ensures that each network layer in the GNN (see Fig. 2) performs a number of matrix multiplications that scales linearly with the number of nodes, i.e., linearly with the number of stabilizer measurements and the overall error rate. We have trained this decoder on simulated data for the surface code using Stim [31] as well as real experimental data on the repetition code [28]. For both of these, the decoder is on par with, or outperforms, state-of-the-art matching decoders [32,33], suggesting that with sufficient data and a suitable neural network architecture, model-free machine learning based decoders trained on experimental data can be competitive for future implementations of quantum error-correcting stabilizer codes.

II. STABILIZER CODES AND DECODING

A stabilizer code is defined through a set of commuting operators constructed from products of Pauli operators acting on a Hilbert space of n data qubits [3]. With n_S independent stabilizers the Hilbert space is split into sectors of dimension 2^{n-n_S}, specified by the parity under each stabilizer. For concreteness we will consider the case $n_S = n - 1$, such that each of the sectors represent a single qubit degree of freedom. Each syndrome measurement is performed with the help of an ancilla qubit following a small entangling circuit with the relevant data qubits. The measured state of the ancilla qubits provide a syndrome $S = \{s_i, i = 1, ..., n_S \mid \in 0, 1\}$, and projects the density matrix of the n qubit state into a single 2-dimensional block, a Pauli frame [34,35]. Given uncertainties in the measurements, a number of rounds are typically performed before the information is interpreted by means of a decoder.

Defining a pair of anticommuting operators Z_L and X_L that commute with the stabilizer group, provides the logical computational space through $Z_L|0\rangle_L = |0\rangle_L$ and $|1\rangle_L = X_L|0\rangle_L$. Assuming a fixed pair of logical operators for a given code defines the corresponding logical states in each Pauli frame. Thus, a number of subsequent rounds of stabilizer measurements, during which the code is affected by decoherence, transforms the density matrix from the initial state

$$\rho = \sum_{i,j\in\{0,1\}} \rho_{ij} |i\rangle_L \langle j|_L \qquad (1)$$



Z_L
$\begin{bmatrix}0\\1\\4\\5\\3\end{bmatrix}$
$\begin{bmatrix}1\\0\\1\\1\\2\end{bmatrix}$
$\begin{bmatrix}1\\0\\2\\2\\2\end{bmatrix}$
$\begin{bmatrix}0\\1\\4\\1\\2\end{bmatrix}$
$\begin{bmatrix}0\\1\\1\\2\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\\4\\5\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\\4\\1\\1\end{bmatrix}$
Z_L
X_L
t
y
x

FIG. 1. Memory experiment on the distance $d = 5$ surface code. Data qubit initialization is followed by $d_t = 2$ stabilizer measurement rounds and a final data qubit measurement round. Data qubits are on the vertices of plaquettes (circles, shown in the bottom and top planes). Ancilla qubits (not shown) at the center of plaquettes provide stabilizer measurements outcomes. The detector graph has nodes corresponding to changes in stabilizers from the previous time step. (Not all edges shown.) Nodes are annotated by the type of stabilizer and the space-time coordinate. The label, here $\lambda_Z = 1$, corresponding to a change of $\langle Z_L\rangle$, measured along the northwest edge. Also shown, bottom layer, are some example stabilizers, and the logical X_L (not measured).

to the final state

$$\rho' = \sum_{i,j\in\{0,1\}} \rho'_{ij} |i\rangle'_L \langle j|'_L \,, \qquad (2)$$

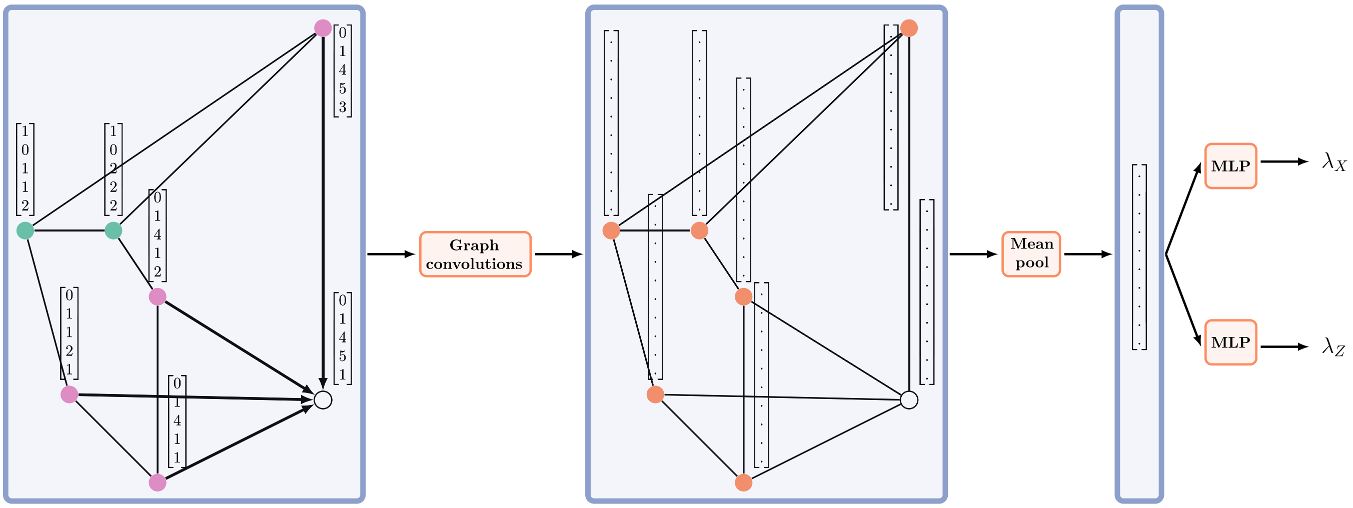


FIG. 2. Schematic of the GNN decoder. It takes as input an annotated detector graph, cf. Fig. 1. Several layers of graph convolutional operations [following Eq. (7)] transform each node feature vector. (The empty circle shows the message passing to this particular node from neighboring nodes on the graph.) Next, a mean-pooling operation averages all the node feature vectors into a single graph embedding, which is independent of the size of the graph. Finally, the latter is passed through two separate dense networks to give two binary class predictors, corresponding to the logical X and Z labels, respectively. (For details, see Appendix A.)

where $|0/1\rangle_L$ ($|0/1\rangle'_L$) are the logical qubit states in the initial (final) Pauli frame. The logical error channel is approximated by

$$\begin{aligned}\rho \to \rho' &= \epsilon_L(\rho)\\ &= (1-P)\tilde{\rho} + P_X X_L \tilde{\rho} X_L + P_Z Z_L \tilde{\rho} Z_L + P_Y Y_L \tilde{\rho} Y_L\,, \qquad (3)\end{aligned}$$

with $Y_L = -iZ_L X_L$ and $P = \sum_{i=X,Y,Z} P_i$. Here $\tilde{\rho} = C(s,s')\rho C(s,s')$, where $C(s,s')|0/1\rangle_L = |0/1\rangle'_L$, is an arbitrary Pauli string that effectuates the change of Pauli frame and commutes with the logical operators. In general there may be additional nonsymmetric channels (see for example [19]), but we will assume that the data (as in [28]) does not resolve such channels.

The probabilities of logical error, P_i, will be quantified by the complete set of syndrome measurements and depend on single and multi-qubit error channels as well as measurement and reset errors. It is the task of the decoder to quantify these in order to maximize the effectiveness of the error correction. Traditionally this is done through computational algorithms that use a specific error model. The framework that most decoders are based on uses independent and identically distributed symmetric noise acting on individual qubits, possibly, for circuit-level noise, complemented by two-qubit gate errors, faulty measurements and ancilla qubit reset errors. Maximum-likelihood decoders [36–41] aim to explicitly account for all possible error configurations that are consistent with the measured syndromes, with their respective probabilities given by the assumed error model. The full set of error configurations fall in four different cosets that map to each other by the logical operators of the code, thus directly providing an estimate of the probabilities P_i that is limited only by the approximations involved in the calculation and the error model. Even though such decoders may be useful for benchmarking and optimizing the theoretical performance of stabilizer codes [28], they are computationally too demanding for real time operation, even for small codes.

An alternative type of decoder is based on the minimum weight perfect matching (MWPM) algorithm [10,42–46]. The objective is to find the single, most likely, configuration of errors consistent with the set of measured stabilizers. Detectors are mapped to nodes of a graph with edges that are weighted by the (negative log) probability of the pair of nodes. For codes where nodes appear in pairs (such as the repetition or surface code), the most likely error corresponds to pairwise matching such that the total weight of the edges is minimized. This algorithm is fast, in practice scaling approximately linearly with the size of the graph. Nevertheless, it has several short-comings that limits accuracy and applicability: 1) Approximate handling of crossing edges (such as coinciding X and Z errors) means that the effective error model is oversimplified. 2) Degeneracies of less likely error configurations are ignored. 3) For models where a single error may give rise to more than two detector events, more sophisticated algorithms are needed [47–53]. These shortcomings can be partially addressed by more sophisticated approaches such as counting multiplicity or using belief propagation [33,54–56], but often at the cost of added computational complexity. Other examples of decoder algorithms are based on decoding from small to large scale, such as cellular-automata [57–59], renormalization group [60], or union-find [49,61]. The latter, in particular, is very efficient, but at the cost of sub-optimal performance.

A. Related work

A number of different deep learning based decoder algorithms have also been formulated, based on supervised learning, reinforcement learning, and genetic neural algorithms [62–82]. Focusing on the works on the surface code and based on supervised learning, these can roughly be separated according to whether they primarily consider perfect stabilizers [62–64,71,76,77,80], or include measurement noise or circuit-level noise [65,68,79,81,82], and whether they are purely data-driven [62,64,65,68,80,82] or involve some auxiliary, model-informed, algorithm or multi-step reduction of decoding [63,71,76,77,79,81].

The present work is in the category, realistic (circuit-level) noise, and purely data-driven. It is distinguished primarily in that we: 1) Use graph neural networks and graph structured data, and 2) Train and test the neural network decoder on real experimental data. In addition, as in several of the earlier works [65,67,68], we emphasize the use of a model-free, purely data-driven, approach. By using experimental stabilizer data, the approximations of traditional model-based decoder algorithms can be avoided. The fact that the real error channels at the qubit level may be asymmetric, due to amplitude damping, have long-range correlations, or involve leakage outside the computational space, is intrinsic to the data. This is also in contrast to other data-driven approaches [21,28,83–85] that use stabilizer data to learn the detailed Pauli channels, optimize a decoder algorithm through the edge weights of a matching decoder, or the individual qubit and measure error rates of a tensor network based decoder, as these are all constrained by a specific error model.

B. Repetition code and surface code

The decoder formalism that we present in this work can be applied to any stabilizer code, requiring only a dataset of measured (or simulated) stabilizers, together with the logical outcomes. Nevertheless, to keep to the core issues of training and performance we consider only two standard scalable stabilizer codes: the repetition code and the surface code.

The bit-flip detecting repetition code is defined on a one-dimensional grid of qubits with neighboring pair-wise $Z_i \otimes Z_{i+1}$ stabilizers. In the Pauli frame with all $+1$ stabilizers, the code words are $|0\rangle_L = |0\rangle^{\otimes n}$ and $|1\rangle_L = |1\rangle^{\otimes n}$. Consider a logical qubit state $|\psi\rangle = \alpha|0\rangle_L + \beta|1\rangle_L$, with complex amplitudes $|\alpha|^2 + |\beta|^2 = 1$. The logical bit-flip operator is given by $X_L = \bigotimes_i X_i$, which sets the code distance $d_X = n$. Assuming perfect stabilizer measurements and independent and identically distributed single qubit bit-flip error probabilities, decoding the repetition code is trivial. For any set of stabilizer violations, i.e., odd parity outcomes, there are only two consistent configurations of errors that map to each other by acting with X_L. A decoder (maximum-likelihood in the case of this simple error model) would suggest the one with fewer errors. The repetition code, set up to detect bit-flip errors, is insensitive to phase flip errors, as is clear from the fact that a phase-flip (Z) error on a single qubit also gives a phase-flip error ($\beta \rightarrow -\beta$) on the logical qubit, corresponding to a code distance $d_Z = 1$. To detect and correct both bit- and phase-flip errors, we need a more potent code, the most promising of which may be the surface code.

We consider the qubit-efficient "rotated" surface code [86–88] (see Fig. 1), constructed from weight-4, $Z^{\otimes 4}$, and $X^{\otimes 4}$, stabilizers (formally stabilizer generators), with complementary weight-2 stabilizers on the boundary. On a square grid of $d \times d$ data qubits, the $d^2 - 1$ stabilizers give one logical qubit. We define the logical operator X_L as a string of X's on the southwest edge, and a string of Z's on the northwest edge, as shown in Fig. 1. These are the two (unique up to products of stabilizers) lowest-weight operators that commute with the stabilizer group, without being part of said group.

Stabilizer measurements are performed by means of entangling circuits between the data qubits and an ancilla qubit. Assuming hardware with one ancilla qubit per stabilizer, and the appropriate gate schedule, these can all be measured simultaneously, corresponding to one round of stabilizer measurements.

C. Memory experiments on the surface code

To train and test our decoder we consider a real or simulated experimental setup, illustrated schematically in Fig. 1, to benchmark a surface code as a quantum memory. The following procedure can be used for any stabilizer code:

(i) Initialize the individual qubits: Data qubits in a fixed or random configuration in the computational basis $|0\rangle$ and $|1\rangle$. Ancilla qubits in $|0\rangle$. The initial data qubit configuration is viewed as a 0′th round of measurements that initialize the Z-stabilizers $s_{Z,i,t=0}$. This also corresponds to an effective measurement $\langle Z_L\rangle_{t=0} = \prod_{i\in Z_L} Z_i = \pm 1$. (Northwest row of qubits in Fig. 1.)

(ii) A first round, $t = 1$, of actual stabilizer measurements is performed, with outcomes $s_{Z,i,t=1}$ and $s_{X,i,t=1}$. This provides the first round of Z-detectors corresponding to changes in $s_{Z,i}$ from the inititalization step. The X-stabilizers $s_{X,i,t=1}$ have randomized outcome, projecting to an even or odd parity state over the four (or two) qubits in the Hadamard ($|+\rangle$, $|-\rangle$) basis. The value of these stabilizers form the reference for subsequent error detecting measurements of the X-stabilizers. Ancilla qubits are reset to 0 after this and subsequent rounds.

(iii) Subsequent rounds $t = 2, ..., d_t$ of Z and X stabilizer measurements provide the input for corresponding detectors based on changes from the previous round.

(iv) Finally, data qubits are measured individually in the Z-basis, which provides a final measurement, $\langle Z_L\rangle_{t=d_t+1}$. The measurements also provide Z-stabilizers, which, being calculated from the actual qubit outcomes rather than by measuring an ancilla, are perfect stabilizers by definition.

The outlined experiment provides a single data point $D = (\{V_Z\}, \{V_X\}, \lambda_Z)$ consisting of a set of Z- and X-detectors $\{V_Z\}$ and $\{V_X\}$, together with a logical label λ_Z. The detectors are defined as the nonzero outcomes of

$$V_{Z,i,t} = s_{Z,i,t-1} \oplus s_{Z,i,t} \tag{4}$$

and

$$V_{X,i,t} = s_{X,i,t-1} \oplus s_{X,i,t} \ , \tag{5}$$

i.e., corresponding to a change in a stabilizer measurement from one-time step to the next. In addition to the stabilizer type, each detector is tagged with its space-time coordinate, (x_i, y_i, t), with $0 \leqslant x, y \leqslant d$ and $1 \leqslant t \leqslant d_t \pm 1$ for Z and X detectors respectively. The logical label is given by

$$\lambda_Z = \tfrac{1}{2}|\langle Z_L\rangle_{t=0} - \langle Z_L\rangle_{t=d_t+1}| \in \{0, 1\}\,. \tag{6}$$

The probability of $\lambda_Z = 1$ is, according to Eq. 3, given by $P_X + P_Y$, and the probability of $\lambda_Z = 0$ by $P_I + P_Z$, corresponding to a logical bit-flip or not.

What has been described is a "memory-Z" experiment [31], i.e., one in which we detect logical bit-flips. Qubits are initialized in the computational basis $|0\rangle$ and $|1\rangle$. A

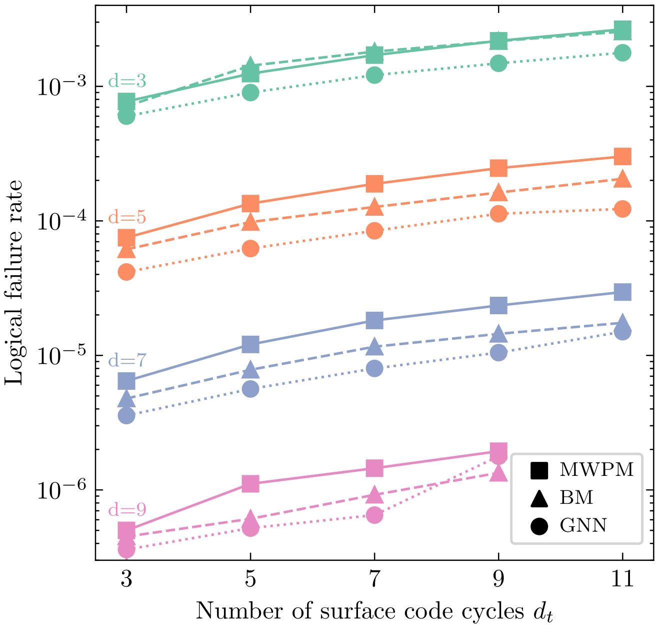


FIG. 3. Logical failure rate versus number of rounds of stabilizer measurements d_t, with simulated circuit-level noise [31] (error rate $p = 1 \cdot 10^{-3}$), on the surface code. Comparing Graph neural network (GNN) decoder to MWPM decoder [32] and belief-matching (BM) decoder [33]. Each data point is evaluated over 10^8 samples (10^7 for $d < 7$). Error bars are smaller than the markers.

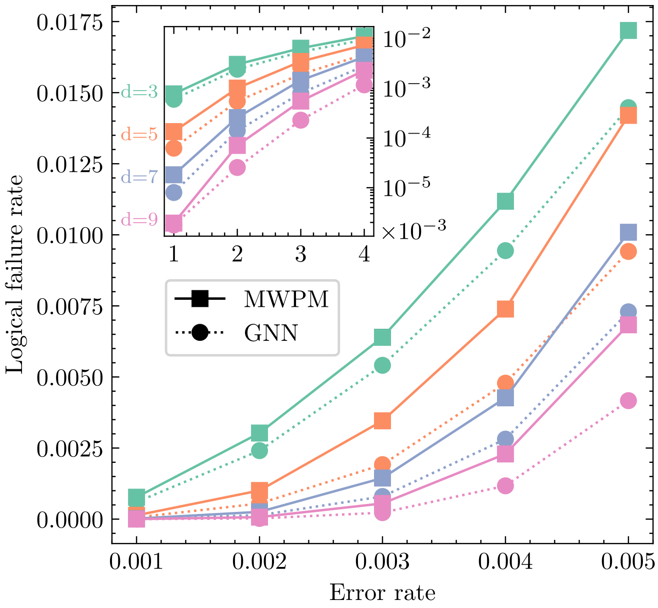


FIG. 4. Logical failure rate versus error rate p, with simulated circuit-level noise, on the surface code with code distance d and $d_t = d$ stabilizer measurement cycles. Else as in Fig. 3.

"memory-X" experiment prepares the qubits in the Hadamard basis, with the role of X- and Z-stabilizers reversed. Physically, in the laboratory, one cannot do both experiments in the same run, as Z_L and X_L do not commute. This also implies that each data point only has one of the two binary labels, λ_Z or λ_X, even though there is information in the detectors about both labels. The neural network will be constructed to predict both labels for a given set of detectors, which implies that the learning framework is effectively that of semi-supervised learning, with partially labeled data. Thus, in contrast to a matching based decoder, which breaks the surface code detectors into two independent sets with a corresponding graph for each, the GNN decoder can make use of the complete information. This, in addition to the fact that it is not constrained by the limitations of the matching algorithm itself, provides a possible advantage in terms of prediction accuracy.

We have also assumed that there is no post-processing to remove leakage. Assuming there is some mechanism of relaxation back to the computational qubit subspace, including the last round of measurements, leakage events will be be handled automatically by the neural network decoder, based on the signature they leave in the detector data.

III. GRAPH NEURAL NETWORK DECODER

A graph neural network (GNN) can be viewed as a trainable message passing algorithm, where information is passed between the nodes through the edges of the graph and processed through a neural network [89–91]. The input is data in the form of a graph $G = (V, E)$, with a set of nodes $V = \{i \,|\, i = 1, .., N\}$ and edges $E = \{(i, j) \,|\, i \neq j \in V\}$, which is annotated by n-dimensional node feature vectors $\vec{X}_i$ and edge weights e_{ij}. The data flow for our GNN-implementation is outlined in Fig. 2, with input in the form of an annotated detector graph and output in the form of two binary predictions. The basic building blocks are the message passing graph convolutional layers which take a graph as input and output an isomorphic graph with transformed feature vectors. Specifically, in this work we have used a standard graph convolution [92], where for each node i the d_{in}-dimensional feature vector $\vec{X}_i$ is transformed to new feature vector $\vec{X}_i'$ with dimension d_{out} according to

$$\vec{X}_i' = \sigma\left(W_1\vec{X}_i + W_2\sum_j e_{ij}\vec{X}_j + \vec{b}\right), \qquad (7)$$

where nonexistent edges are indicated by $e_{ij} = 0$. Here W_1 and W_2 are $d_{\text{out}} \times d_{\text{in}}$ dimensional trainable weight matrices, $\vec{b}$ is a d_{out}-dimensional trainable bias vector. The nonlinear activation function, σ, acts element-wise, outputting the new feature vector. A standard form, used in this work, is the rectified linear unit, $\sigma(x) = \text{ReLU}(x) = \max(0, x)$.

For the task at hand, which is graph classification, a number of subsequent graph convolutions are followed by a pooling layer that contracts the information to a single vector, a graph embedding, which is independent of the dimension of the graph. In this work, we use a simple mean-pooling layer

$$\vec{X}_{\text{mean}} = N^{-1}\sum_i \vec{X}_i, \qquad (8)$$

where N is the number of nodes in the graph. For the classification we use two structurally identical, but independent, multi-layer perceptrons (MLP), i.e. standard dense feed-forward neural networks, where each layer acts acts as

$$\vec{X}' = \sigma(W\vec{X} + \vec{b}), \qquad (9)$$

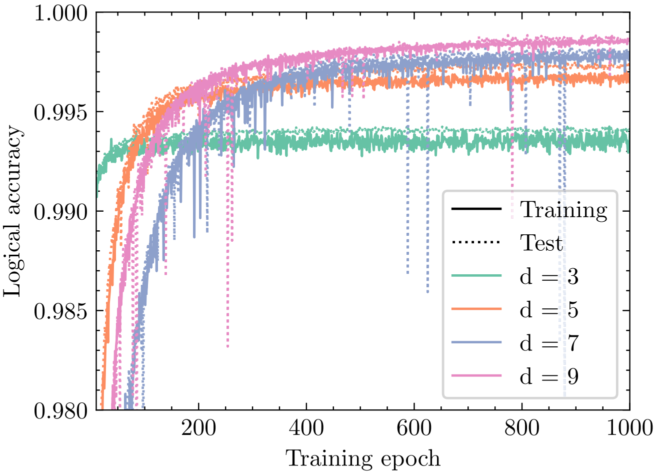


FIG. 5. GNN training and test accuracy versus number of training epochs for circuit-level noise, comparing different code distances ($d_t = d$). Each epoch amounts to training on a freshly generated dataset with 10^7 samples using an error rate randomly selected from $p = [0.001, 0.002, ..., 0.005]$. The test set is a fixed dataset of the same type containing $5 \cdot 10^4$ data points. The discrepancy with accuracies in Fig. 3 is due to the exclusion of empty graphs (trivial syndromes) from the training data.

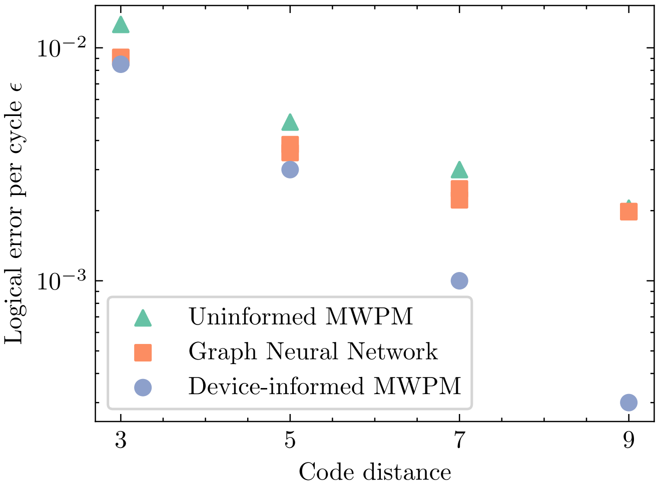


FIG. 6. Decoding experimental data [28] on the repetition code with code distance d, over 50 rounds of stabilizer measurements. Comparing GNN decoder, using a dataset containing $(26 - d) \cdot 5 \cdot 10^7$ graphs, with a MWPM decoder with "device-optimized" edge weights [28] and a simple model-free MWPM decoder with 1-norm edge weights. The training-test split of the dataset is 99 to 1, and the logical failure rate is mapped to an error rate per round. Results for two different random training-test splits are shown.

with a trainable weight matrix W and bias vector $\vec{b}$. The input to the two MLPs is the pooled output from the graph convolution layers, $\vec{X}_{\text{mean}}$. Each MLP ends with a single node with sigmoid activation, $\sigma(x) = 1/(1 + e^{-x}) \in [0, 1]$, that acts as a binary classifier. The weights and biases of the complete network are trained using stochastic gradient descent with a loss function which is a sum of the binary cross entropy loss of the network output with respect to the binary labels. Since the experimental data, or simulated data, only has one of the two binary labels (λ_Z, λ_X) for each complete detector graph, gradients are only calculated for the provided label.

To avoid overfitting to the training data we employ two different approaches depending on the amount of available data. In using experimental data from [28], we use a two-way split into a training set and a test set. To avoid diminishing the training data further, we do not use a validation set, and instead train for a fixed number of epochs. We observe (see Fig. 7) that the test accuracy does not change significantly over a large number of epochs, even though the network continues to overfits.

For the case with simulated data (Fig. 5), we avoid overfitting by not reusing data. Each batch of the training data consists of freshly generated, labeled detector graphs. A fixed test set is used to gauge the performance.

The GNN training and testing is implemented in PyTorch Geometric [93], simulated data is generated using Stim [31], the MWPM decoding results use PyMatching [32,94], and the belief-matching results uses the code provided with [33]. The Adam optimizer is used for stochastic gradient descent, using manual learning rate decrements when the training accuracy has leveled out. Details on the training procedure can be found in Appendix A. Several other graph layers were experimented with, including graph attention for both convolutions [95] and pooling [96,97], as well as top$_k$ pooling [98,99]. These were found not to improve results. The width and depth of the final network was arrived at after several rounds of iterations, but no systematic ablation studies were done. We expect that larger code distances, i.e., larger graphs, will require scaling up the network, following the increased complexity of the decoding problem. We use a fixed-size network for $d \leqslant 7$, and a somewhat-larger network for $d = 9$ (see also Sec. IV D).

A. Data structure

As discussed previously, the data is in a form $D = (\{V_Z\}, \{V_X\}, \lambda_{Z/X})$, consisting of a set of detectors $V_{Z/X}$, specified by a space-time coordinate, together with a binary label. Based on this, we construct a single graph. Each node corresponds to a detector event, and is annotated by a 5-vector (for the surface code with circuit-level noise) $\vec{X} = (b_1, b_2, x, y, t)$ containing the space-time coordinate (x, y, t) and two exclusive binary (one-hot encoded) labels with $\vec{b} = (1, 0)$ for an X-stabilizer and $\vec{b} = (0, 1)$ for a Z-stabilizer. (The encoding of the type of stabilizer may be superfluous, as it can be deduced from the coordinate.) We initially consider a complete graph, with edge weights given by the euclidean distance between the detectors, $e_{ij} = 1/\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (t_i - t_j)^2}$. The edge weights give a rough measure of the likelihood that two detectors are triggered due to the same error or set of errors and are used to prune edges in the graph. Lower weight edges are removed, leaving only a fixed maximal node degree, reducing the size of each data point such that it grows linearly with the number of nodes. The pruning using euclidean distance is efficiently implemented in the integrated data generation and training pipeline as a data-preprocessing step.

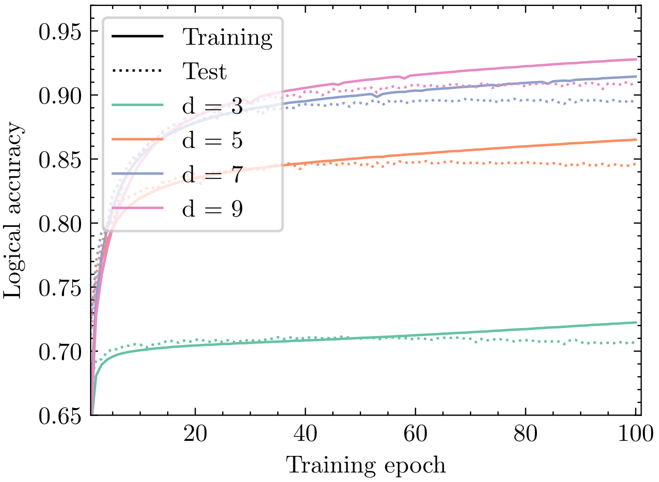


FIG. 7. Training curves for the GNN decoder on repetition code, following Fig. 6. Each epoch trains through the whole training set, which eventually leads to overfitting, where the training accuracy starts to significantly surpass the test accuracy. To maximize the amount of training data, no validation set was used. No early stopping was implemented in order to avoid optimizing results to the test set.

IV. RESULTS

The GNN-based decoder has been implemented, trained, and tested on the surface code and the repetition code. The main focus is on using simulated data or experimental data, presented in Secs. IV A and IV B, respectively. We also present some results on the surface code with perfect stabilizers, Sec. IV C, where we are able to train the network for larger code distances.

A. Surface code with circuit-level noise

We use Stim to generate data with circuit-level noise. Simulated circuits use standard settings for the surface code, containing Hadamard single qubit gates, controlled-Z (CZ) entangling gates, and measure and reset operations. All of these operations, and the idling, contain Pauli noise, scaled by an overall error rate p (see Appendix B). Datasets are generated in batches of varying sizes (see Appendix A), that each give a single gradient descent update of the neural network weights. For presentation purposes, the batches are grouped into epochs containing 10^7 data points in total, after which the test accuracy is evaluated. As discussed previously, to eliminate overfitting to the training data, no data is reused. This is feasible as the data generation is very fast.

Figure 3 shows test results evaluated at $p = 1.0 \cdot 10^{-3}$ for decoders trained with data using an even mix of error rates $p = \{1.0, 2.0, 3.0, 4.0, 5.0\} \cdot 10^{-3}$ and memory-Z experiments. The logical failure rate is thus approximately 50% of the true failure rate (up to correlations between failures in X_L and Z_L), but consistent with the type of data that would be experimentally accessible. (We have also tried training and testing with a mix of memory-Z and memory-X experiments, which works as well but takes longer to train to the same accuracy.) The rationale for using larger error rates during training is to include a relatively larger fraction of graphs with many nodes, under the assumption that these will generally be harder to decode. We compare to MWPM and belief-propagation augmented MWPM (belief-matching). Both these decoders use the information provided by the simulated error model to optimize edge weights on the decoding graph, where the BM algorithm additionally propagates information within and between the Z- and X-detector graphs for each instance. Despite the fact that the GNN decoder uses only the data provided by the simulated measurements, we find that with sufficient training the GNN decoder outperforms the matching decoders. For the largest code-distance considered, $d = 9$, a larger network was used (see Appendix A), and the training has not converged to consistently outperform BM for all cycle depths considered. Figure 4 also shows the performance of the GNN under varying error rates versus MWPM. We find that the networks have good performance within the whole range of error rates over which it was trained.

A different network is trained for each code distance d and for each number of rounds of stabilizer measurements d_t. Figure 5 shows a representative plot of the training and test accuracy, evaluated on the mixed error rate dataset. No data is reused, which implies that the network cannot overfit and that the test accuracy closely follows the training accuracy. Further details are given in Appendix A.

B. Repetition code using experimental data

Having trained GNN based decoders on simulated data in the previous section, we now turn to real experimental data. We use the public data provided together with [28]. This contains data on both the $d = 3$ and $d = 5$ surface codes as well as the $d = 25$ bit-error correcting repetition code. All datasets are of the form described in Sec. II C, thus readily transferred to the annotated and labeled graphs used to train the GNN, as described in Sec. III A. The datasets contain approximately 10^6 data points for the different codes, code distances, and varying number of stabilizer rounds.

Our attempts to train a GNN on the data provided for the various implementations of surface code were generally unsuccessful. While it gave good results on the training data, the logical failure rate on the test set was poor. Given the fact that on the order of 10^9 data points were used for the simulated circuit-level noise on the surface code (Sec. IV A), it is not surprising that the significantly smaller dataset turned out to be insufficient. The network cannot achieve high accuracy without overfitting to the training data given the relatively small dataset.

For the repetition code, the data which is provided is of a single type, for a $d = 25$ code measured over $d_t = 50$ rounds. Each round thus contains the measurement of 24 ancilla qubits for the ZZ stabilizers of the two neighboring data qubits along a one-dimensional path. As done in [28] this data can be split up into data for smaller repetition codes, by simply restricting to stabilizers over a subset of d subsequent data qubits. In this way the dataset can be increased by a factor $25 - (d - 1)$, and used to train a single GNN for each code distance. It should be noted that this is suboptimal, compared to generating the same amount of data on single distance d device, as variations in the performance of the constituent qubits and gates will be averaged out in the dataset. Nevertheless, using this scheme we successfully trained GNN decoders for short distance rep-

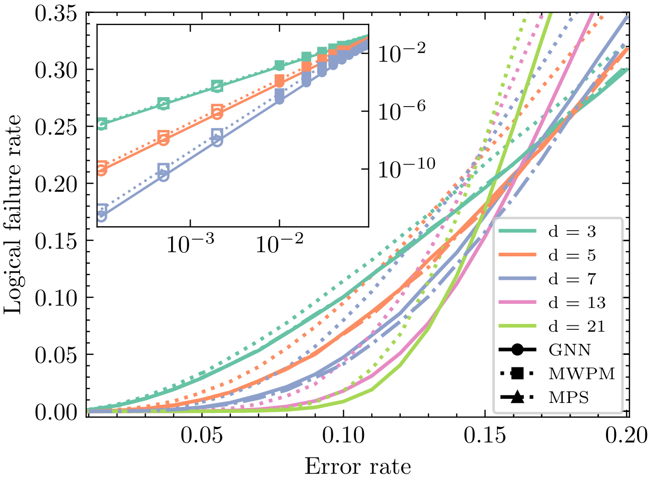


FIG. 8. Decoding the rotated surface code with perfect stabilizers and code distance d. Logical failure rate versus error rate p, for depolarizing noise, evaluated over failures with respect to both $X_L(\lambda_X)$ and $Z_L(\lambda_Z)$. Comparing the GNN decoder with MWPM decoder that has full information of the data-generating error model. Each data point is evaluated over 10^5 data points (10^8 for $p < 10^{-2}$). Dashed lines is the accuracy using a matrix product state (MPS) decoder [100] at code distances 3 to 7. Inset shows low-p failure rates for $d \leqslant 7$, where open markers are based on sampling only the lowest weight errors that fail.

etition codes, with test accuracies shown in Fig. 6. Results for (what we refer to as) "Device-optimized MWPM" is taken from [28]. The GNN decoder performs almost on par with this sophisticated matching decoder for $d = 3$. As expected, the relative performance deteriorates with increased code distance. We expect that we would need more training data for larger code distance, but instead we have access to less.

As the comparison with the matching decoder that uses a device specific error model may be biased compared to using training data from different devices, as mentioned above, we also give results for an "uniformed" matching decoder with edge weights based on the 1-norm distance between spacetime coordinates. It may also be noted that using MWPM corresponds to a near optimal decoder for the repetition code, at least for the case of phenomenological measurement noise where it is equivalent to bit-flip error surface code. This is in contrast to the surface code, for which MWPM is suboptimal, even in the case of perfect stabilizers. Thus, outperforming MWPM for the repetition code may be more challenging than for the surface code.

C. Surface code with perfect stabilizers

To complement the results on circuit-level noise and experimental data we have also trained the GNN decoder on the surface code with perfect stabilizers under depolarizing noise. The same network (see Appendix A) is used as for circuit-level noise, but trained at $p = [0.01, 0.05, 0.1, 0.15]$. For this problem, both labels, corresponding to logical bit-and/or phase-flips, are used for training and testing.

Results up to code distance $d = 21$ are shown in Fig. 8 and found to significantly outperform MWPM. We also compare to a tensor network based [100] maximum likelihood decoder (MLD), showing that for code distance $d \leqslant 5$ the GNN decoder has converged to the level of being an approximate MLD. For very low error rates and larger code-distances $d > 7$, we find that the networks still fail for some low weight errors that should be correctable, making the asymptotic behavior worse than MWPM. Nevertheless, we expect that more training, and training tailored to low error rates, could resolve this.

We do not attempt to derive any threshold for the GNN decoder. Given a sufficiently expressive network we expect that the decoder would eventually converge to a maximum likelihood decoder, but in practice the accuracy is limited by the training time. It gets progressively more difficult to converge the training for larger code distances, which means that any threshold estimate will be a function of the training time versus code distance. In fact, in principle, since the threshold is a $d \to \infty$ quantity, we would not expect that a supervised learning algorithm can give a proper threshold if trained separately for each code distance, as is done in this work. Using GNN's (as opposed to a network acting on a fixed-size grid) it is, in principle, quite natural to use the same network to decode any distance code, as the data objects (detector graphs) have the same structure. We have investigated training the same network for different code distances and different number of rounds. This shows some promise, but has not achieve accuracy levels that can match MWPM.

D. Scalability

We are limited to relatively small codes in this work. For the repetition code using experimental data, it is quite clear that main limitation to scaling up the code distance is the size of the available dataset. For the surface code using simulated data it is challenging to increase the code distance while still surpassing MWPM. As the logical failure rates decrease exponentially with code distance, the test accuracy of the supervised training needs to follow. One way to counter this is to increase the number of stabilizer cycles, d_t, but this also increases the graph size, making the training more challenging from the perspective of increased memory requirements as well as the increased complexity of the data.

Nevertheless, it is interesting to explore the intrinsic scalability of the algorithm, by quantifying how the decoding time using a fixed size GNN scales with the code size. In Fig. 9 we present results on the decoding time per syndrome for the surface code, as a function of code volume d^2d_t, at fixed error rate, compared to PyMatching 2 [94]. The network is fixed to the smaller network described in Appendix A. In line with expectations, the GNN inference scales approximately linearly with the code volume, i.e., average graph size, $T \sim d^2d_t$. The number of matrix operations per graph convolutional layer, following Eq. (7), is proportional to the number of nodes multiplied by the number of edges. The number of layers is fixed, multiplying this by a constant factor. The feature vector pooling is proportional to the number of nodes, whereas the subsequent dense network classifiers are independent of the graph size. We find that inference scales slightly better than the highly optimized matching decoder. However, several caveats are in order. 1) The size of the GNN is fixed. Larger

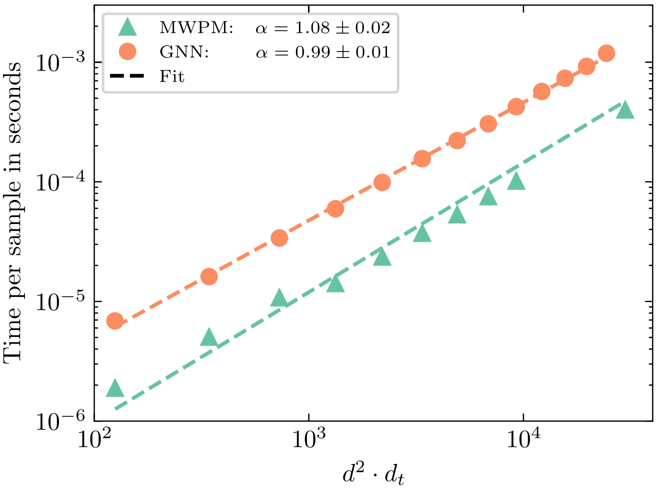


FIG. 9. Scaling of average decoding time per syndrome versus code volume $d^2 d_t$ for GNN and MWPM using PyMatching 2 [94]. Circuit level noise sampled at $p = 10^{-3}$. Dotted lines show a regression according to the ansatz: $T = C \cdot (d^2 \cdot d_t)^{\alpha}$, with the GNN showing sub-linear scaling.

code will require scaled-up networks, unless the error rate is scaled down accordingly; 2) The network has not been trained on code distances larger than $d = 9$. It is only a test of the decoding time, not the accuracy. 3) Data (for both algorithms) is batched for fast inference. Treating batched data doesn't seem viable for real time decoding. Moving to other types of hardware (see, e.g., [101]), such as field-programmable gate arrays (FPGA) or an application-specific integrated circuits (ASIC), will be necessary for real-time decoding of superconducting devices, requiring μs per cycle decoding times, using neural networks.

V. CONCLUSION AND OUTLOOK

In this paper we develop a model-free, data-driven, approach to decoding quantum error correcting stabilizer codes, using graph neural networks. A real or simulated memory experiment is represented as a single detector graph, with annotated nodes corresponding to the type of stabilizer and its space-time coordinate, and a binary label corresponding to whether or not a logic bit-flip has occurred. The maximal node degree is capped by cropping edges between distant nodes. The data is used to train a convolutional GNN for graph classification, with classes corresponding to logical Pauli operations, and used for decoding. We show that we can use experimental and simulated data, for the repetition code and surface code respectively, to train a decoder with logical failure rates on par with minimum weight perfect matching, despite the latter having detailed information about the underlying real or simulated error channels. The use of a graph structure provides an efficient way to store and process the syndrome data. Training the GNN requires significant amounts of training data, but as shown in the case of simulations, data can be produced in parallel with training. Network inference, i.e., using the network as a decoder, is fast, scaling approximately linearly with the space-time dimension of the code.

As an extension of this work there are several interesting possibilities to explore. One example is to use a GNN for edge-weight generation within a hybrid algorithm with a matching decoder (similar to [21]). This would depart from the pure data-driven approach pursued in this paper, with peak performance limited by the matching decoder, but with the potential advantage of requiring less data to train. An alternative to this, to potentially improve performance and lower data requirements, is to use device specific input into edge weights, or encode soft information on measurement fidelities into edge or node features.

Going beyond the most standard codes considered in this paper, we expect that any error correcting code for which labeled detector data can be generated can also be decoded with a GNN. This includes Clifford-deformed stabilizer codes [102–106], color codes [107,108], hexagonal stabilizer codes [109–112], and quantum low-density parity check (LDPC) codes [113–115], where syndrome defects are not created in pairs, but potentially also Floquet type codes [116,117]. In addition, heterogeneous and correlated noise models [118,119] would also be interesting to explore, where in particular the latter is difficult to handle with most standard decoders. Adding soft information, e.g., the estimated reliability of a stabilizer measurement, is also a natural next step for this type of decoder [120]. The software code for the project can be found at [121].

Shortly after posting the preprint for this paper, related work, Varbanov *et al.* [122], using a recurrent neural network architecture was presented. That network was trained on simulated data and tested on experimental data [28], but did not implement training exclusively on experimental data. Related work has subsequently been presented in Bausch *et al.* [123], using a transformer-augmented recurrent architecture.

ACKNOWLEDGMENTS

We acknowledge financial support from the Knut and Alice Wallenberg Foundation through the Wallenberg Centre for Quantum Technology (WACQT), and the Marianne and Marcus Wallenberg foundation through EDU-WACQT. Computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at Chalmers Centre for Computational Science and Engineering (C3SE), partially funded by the Swedish Research Council through Grant Agreements No. 2022-06725 and No. 2018-05973, as well as by the Berzelius resource provided by the Knut and Alice Wallenberg Foundation at the National Supercomputer Centre. We thank Viktor Rehnberg and Hampus Linander for input on the neural network training. This work was also supported by the Dutch National Growth Fund (NGF) as part of the Quantum Delta NL programme.

APPENDIX A: GNN ARCHITECTURE AND TRAINING

Figure 10 displays the architecture of the GNN decoder. The node features are sent through seven subsequent graph convolutional layers [Eq. (7)]. The node features are passed through a rectified linear unit (ReLU) activation function

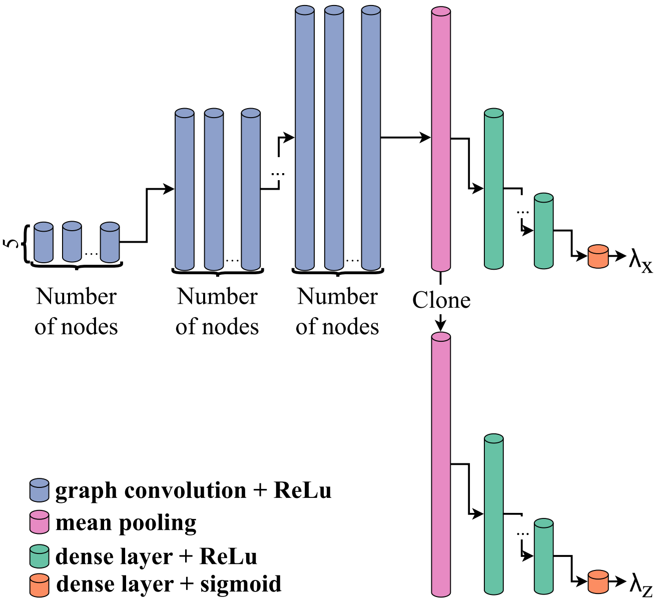


FIG. 10. Schematic of the GNN architecture, with details in Table I. The same architecture is used for all the results, except that for the repetition code there is only one output head. The input dimension is two (2D space-time coordinate) for the repetition code and four (two types of stabilizers, and 2D spatial coordinate) for the surface code with perfect stabilizers.

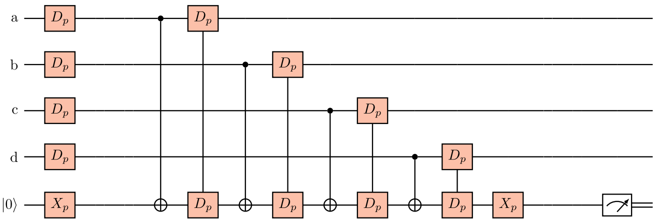


FIG. 11. Quantum circuit for measuring the weight-four stabilizer Z_{abcd} under circuit-level noise.

(which corresponds to chopping negative values) after each layer. After the graph convolutional layers, the node features from all nodes are pooled into one high-dimensional vector by computing the mean across all nodes. This vector is then cloned and sent to two identical fully connected neural networks. Both heads consist of several dense layers, which map the pooled node feature vector down to one real-valued number which is mapped to the range 0 to 1 through a sigmoid function. The input and output dimension d_{in} and d_{out} of the graph convolutional and dense layers can be found in Table I.

Networks are trained on NVIDIA Tesla A100 HGX GPU's using the pytorch geometric knn module to generate graphs in parallel. For gradient descent, samples are batched in batches of sizes ranging from $6 \cdot 10^3$ for the biggest code instances ($d = d_t = 9$) to $26 \cdot 10^3$ for the smallest code instances ($d = d_t = 3$). The batch sizes are chosen to fully utilize the GPU during training. The learning rate is set to 10^{-4} and decreased manually to 10^{-5}, whenever the validation accuracy reached a plateau. The training scripts, trained models and details on all hyper-parameters are available at [121].

TABLE I. Input and output dimensions of each layer in the GNN decoder. Left: $d \leqslant 7$, total parameters: 1.36×10^6. Right: $d = 9$, total parameters: 2.35×10^6.

	$d \leqslant 7$		$d = 9$	
Layer	d_{in}	d_{out}	d_{in}	d_{out}
GraphConv$_1$	5	32	5	32
GraphConv$_2$	32	128	32	128
GraphConv$_3$	128	256	128	256
GraphConv$_4$	256	512	256	512
GraphConv$_5$	512	512	512	512
GraphConv$_6$	512	256	512	512
GraphConv$_7$	256	256	512	512
Dense$_1$	256	256	512	512
Dense$_2$	256	128	512	256
Dense$_3$	128	64	256	128
Dense$_4$	64	1	128	64
Dense$_5$	–	–	64	32
Dense$_6$	–	–	32	16
Dense$_7$	–	–	16	1

APPENDIX B: STABILIZER CIRCUITS AND ERROR MODEL FOR CIRCUIT-LEVEL NOISE

Quantum circuits for weight-four Z- (X-) stabilizers of the surface code are displayed in Fig. 11 (12). The gate set used for the stabilizer measurements consists of the Hadamard gate (H), and the $CNOT$ gate. Under circuit-level noise, single-qubit depolarizing noise gate D_p (which applies gate $\sigma_i, i \in \{X, Y, Z\}$ where any of the gates is applied with probability $p/3$, and I with probability $1 - p$) acts on the data qubits before each stabilizer measurement cycle and on each target qubit after single-qubit gates. Two-qubit depolarizing noise gates (which apply gate $\sigma_i\sigma_j, i, j \in \{I, X, Y, Z\}$, where II is acted on with probability $1 - p$, and the rest with probability $p/15$) act on the two qubits involved after every two-qubit gate. Furthermore, each qubit suffers from reset- and measurement-error with probability p, displayed by operators X_p when measuring and resetting in the computational basis. The precise model is specified in `stim.Circuit.generated(‘‘surface_code: rotated_memory_z’’)`.

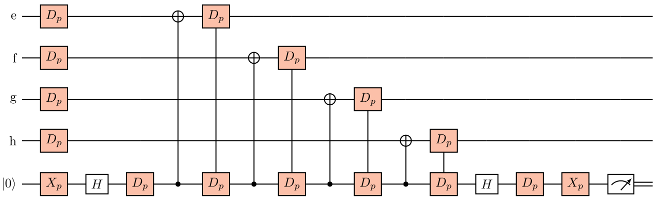


FIG. 12. Quantum circuit for measuring the weight-four stabilizer X_{efgh} under circuit-level noise.

[1] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, Phys. Rev. A **52**, R2493 (1995).

[2] A. M. Steane, Error correcting codes in quantum theory, Phys. Rev. Lett. **77**, 793 (1996).

[3] D. Gottesman, *Stabilizer Codes and Quantum Error Correction* (California Institute of Technology, 1997).

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

[5] S. M. Girvin, Introduction to quantum error correction and fault tolerance, arXiv:2111.08894.

[6] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. van den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, and A. Kandala, Evidence for the utility of quantum computing before fault tolerance, Nature (London) **618**, 500 (2023).

[7] K. Temme, S. Bravyi, and J. M. Gambetta, Error mitigation for short-depth quantum circuits, Phys. Rev. Lett. **119**, 180509 (2017).

[8] Y. Li and S. C. Benjamin, Efficient variational quantum simulator incorporating active error minimization, Phys. Rev. X **7**, 021050 (2017).

[9] S. B. Bravyi and A. Y. Kitaev, Quantum codes on a lattice with boundary, arXiv:quant-ph/9811052.

[10] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, J. Math. Phys. **43**, 4452 (2002).

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

[12] R. Raussendorf and J. Harrington, Fault-tolerant quantum computation with high threshold in two dimensions, Phys. Rev. Lett. **98**, 190504 (2007).

[13] 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).

[14] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, State preservation by repetitive error detection in a superconducting quantum circuit, Nature (London) **519**, 66 (2015).

[15] M. Takita, A. W. Cross, A. D. Córcoles, J. M. Chow, and J. M. Gambetta, Experimental demonstration of fault-tolerant state preparation with superconducting qubits, Phys. Rev. Lett. **119**, 180501 (2017).

[16] J. R. Wootton and D. Loss, Repetition code of 15 qubits, Phys. Rev. A **97**, 052313 (2018).

[17] J. R. Wootton, Benchmarking near-term devices with quantum error correction, Quantum Sci. Technol. **5**, 044004 (2020).

[18] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, Repeated quantum error detection in a surface code, Nat. Phys. **16**, 875 (2020).

[19] K. J. Satzinger *et al.*, Realizing topologically ordered states on a quantum processor, Science **374**, 1237 (2021).

[20] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina, and C. Monroe, Fault-tolerant control of an error-corrected qubit, Nature (London) **598**, 281 (2021).

[21] Z. Chen *et al.*, Exponential suppression of bit or phase errors with cyclic error correction, Nature (London) **595**, 383 (2021).

[22] A. Erhard, H. Poulsen Nautrup, M. Meth, L. Postler, R. Stricker, M. Stadler, V. Negnevitsky, M. Ringbauer, P. Schindler, H. J. Briegel, R. Blatt, N. Friis, and T. Monz, Entangling logical qubits with lattice surgery, Nature (London) **589**, 220 (2021).

[23] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz, Realization of real-time fault-tolerant quantum error correction, Phys. Rev. X **11**, 041058 (2021).

[24] J. F. Marques, B. M. Varbanov, M. S. Moreira, H. Ali, N. Muthusubramanian, C. Zachariadis, F. Battistel, M. Beekman, N. Haider, W. Vlothuizen, A. Bruno, B. M. Terhal, and L. DiCarlo, Logical-qubit operations in an error-detecting surface code, Nat. Phys. **18**, 80 (2022).

[25] L. Postler, S. Heussen, I. Pogorelov, M. Rispler, T. Feldker, M. Meth, C. D. Marciniak, R. Stricker, M. Ringbauer, R. Blatt, P. Schindler, M. Müller, and T. Monz, Demonstration of fault-tolerant universal quantum gate operations, Nature (London) **605**, 675 (2022).

[26] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, C. K. Andersen *et al.*, Realizing repeated quantum error correction in a distance-three surface code, Nature (London) **605**, 669 (2022).

[27] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner, V. Vuletić, and M. D. Lukin, A quantum processor based on coherent transport of entangled atom arrays, Nature (London) **604**, 451 (2022).

[28] Google Quantum AI, Suppressing quantum errors by scaling a surface code logical qubit, Nature (London) **614**, 676 (2023).

[29] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, L. Ascarrunz, C. Barnes, J. Bartolotta, B. Bjork, P. Blanchard, M. Bohn, J. G. Bohnet, N. C. Brown, N. Q. Burdick, W. C. Burton, S. L. Campbell, J.P. Campora, III, C. Carron, J. Chambers, J. W. Chan, Y. H. Chen *et al.*, A race track trapped-ion quantum processor, Phys. Rev. X **13**, 041052 (2023).

[30] N. Sundaresan, T. J. Yoder, Y. Kim, M. Li, E. H. Chen, G. Harper, T. Thorbeck, A. W. Cross, A. D. Córcoles, and M. Takita, Demonstrating multi-round subsystem quantum error correction using matching and maximum likelihood decoders, Nat. Commun. **14**, 2852 (2023).

[31] C. Gidney, Stim: a fast stabilizer circuit simulator, Quantum **5**, 497 (2021).

[32] O. Higgott, Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching, arXiv:2105.13082.

[33] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, Improved decoding of circuit noise and fragile boundaries of tailored surface codes, Phys. Rev. X **13**, 031007 (2023).

[34] E. Knill, Quantum computing with realistically noisy devices, Nature (London) **434**, 39 (2005).

[35] N. de Beaudrap and D. Horsman, The ZX calculus is a language for surface code lattice surgery, Quantum **4**, 218 (2020).

[36] J. R. Wootton and D. Loss, High Threshold Error Correction for the Surface Code, Phys. Rev. Lett. **109**, 160503 (2012).

[37] A. Hutter, J. R. Wootton, and D. Loss, Efficient Markov chain Monte Carlo algorithm for the surface code, Phys. Rev. A **89**, 022326 (2014).

[38] S. Bravyi, M. Suchara, and A. Vargo, Efficient algorithms for maximum likelihood decoding in the surface code, Phys. Rev. A **90**, 032326 (2014).

[39] K. Hammar, A. Orekhov, P. W. Hybelius, A. K. Wisakanto, B. Srivastava, A. F. Kockum, and M. Granath, Error-rate-agnostic decoding of topological stabilizer codes, Phys. Rev. A **105**, 042616 (2022).

[40] L. P. Pryadko, On maximum-likelihood decoding with circuit-level errors, Quantum **4**, 304 (2020).

[41] C. T. Chubb, General tensor network decoding of 2D pauli codes, arXiv:2101.04125.

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

[43] D. S. Wang, A. G. Fowler, A. M. Stephens, and L. C. L. Hollenberg, Threshold Error Rates for the Toric and Planar Codes, Quantum Inf. Comput. **10**, 456 (2010).

[44] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg, Surface code quantum computing with error rates over 1%, Phys. Rev. A **83**, 020302(R) (2011).

[45] 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).

[46] B. J. Brown, Conservation laws and quantum error correction: towards a generalised matching decoder, in *IEEE BITS the Information Theory Magazine* (IEEE, 2022), Vol. 2, pp. 5–19.

[47] N. Delfosse, Decoding color codes by projection onto surface codes, Phys. Rev. A **89**, 012317 (2014).

[48] A. M. Stephens, Efficient fault-tolerant decoding of topological color codes, arXiv:1402.3037.

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

[50] D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, Fault-tolerant thresholds for the surface code in excess of 5% under biased noise, Phys. Rev. Lett. **124**, 130501 (2020).

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

[52] L. Berent, L. Burgholzer, P.-J. H. S. Derks, J. Eisert, and R. Wille, Decoding quantum color codes with maxsat Quantum **8**, 1506 (2024).

[53] A. Benhemou, K. Sahay, L. Lao, and B. J. Brown, Minimising surface-code failures using a color-code decoder Quantum **9**, 1632 (2025).

[54] N. Delfosse and J.-P. Tillich, A decoding algorithm for CSS codes using the X/Z correlations, in *2014 IEEE International Symposium on Information Theory* (IEEE, 2014), pp. 1071–1075.

[55] B. Criger and I. Ashraf, Multi-path summation for decoding 2D topological codes, Quantum **2**, 102 (2018).

[56] L. Caune, J. Camps, B. Reid, and E. Campbell, Belief propagation as a partial decoder, arXiv:2306.17142.

[57] M. Herold, E. T. Campbell, J. Eisert, and M. J. Kastoryano, Cellular-automaton decoders for topological quantum memories, npj Quantum Inf. **1**, 15010 (2015).

[58] A. Kubica and J. Preskill, Cellular-automaton decoders with provable thresholds for topological codes, Phys. Rev. Lett. **123**, 020501 (2019).

[59] J. F. S. Miguel, D. J. Williamson, and B. J. Brown, A cellular automaton decoder for a noise-bias tailored color code, Quantum **7**, 940 (2023).

[60] G. Duclos-Cianci and D. Poulin, Fast decoders for topological quantum codes, Phys. Rev. Lett. **104**, 050504 (2010).

[61] S. Huang, M. Newman, and K. R. Brown, Fault-tolerant weighted union-find decoding on the toric code, Phys. Rev. A **102**, 012419 (2020).

[62] G. Torlai and R. G. Melko, Neural Decoder for Topological Codes, Phys. Rev. Lett. **119**, 030501 (2017).

[63] S. Krastanov and L. Jiang, Deep neural network probabilistic decoder for stabilizer codes, Sci. Rep. **7**, 11003 (2017).

[64] S. Varsamopoulos, B. Criger, and K. Bertels, Decoding small surface codes with feedforward neural networks, Quantum Sci. Technol. **3**, 015004 (2018).

[65] P. Baireuther, T. E. O'Brien, B. Tarasinski, and C. W. Beenakker, Machine-learning-assisted correction of correlated qubit errors in a topological code, Quantum **2**, 48 (2018).

[66] N. P. Breuckmann and X. Ni, Scalable Neural Network Decoders for Higher Dimensional Quantum Codes, Quantum **2**, 68 (2018).

[67] 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 J. Phys. **21**, 013003 (2019).

[68] C. Chamberland and P. Ronagh, Deep neural decoders for near term fault-tolerant experiments, Quantum Sci. Technol. **3**, 044002 (2018).

[69] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing Quantum Error Correction Codes with Reinforcement Learning, Quantum **3**, 215 (2019).

[70] N. Maskara, A. Kubica, and T. Jochym-O'Connor, Advantages of versatile neural-network decoding for topological codes, Phys. Rev. A **99**, 052351 (2019).

[71] X. Ni, Neural Network Decoders for Large-Distance 2D Toric Codes, Quantum **4**, 310 (2020).

[72] R. Sweke, M. S. Kesselring, E. P. van Nieuwenburg, and J. Eisert, Reinforcement learning decoders for fault-tolerant quantum computation, Mach. Learn.: Sci. Technol **2**, 025005 (2021).

[73] P. Andreasson, J. Johansson, S. Liljestrand, and M. Granath, Quantum error correction for the toric code using deep reinforcement learning, Quantum **3**, 183 (2019).

[74] L. D. Colomer, M. Skotiniotis, and R. Muñoz-Tapia, Reinforcement learning for optimal error correction of toric codes, Phys. Lett. A **384**, 126353 (2020).

[75] D. Fitzek, M. Eliasson, A. F. Kockum, and M. Granath, Deep Q-learning decoder for depolarizing noise on the toric code, Phys. Rev. Res. **2**, 023230 (2020).

[76] S. Gicev, L. C. Hollenberg, and M. Usman, A scalable and fast artificial neural network syndrome decoder for surface codes, arXiv:2110.05854.

[77] D. Bhoumik, P. Sen, R. Majumdar, S. Sur-Kolay, L. K. K. J, and S. S. Iyengar, Efficient decoding of surface code syndromes for error correction in quantum computing, arXiv:2110.10896.

[78] H. Théveniaut and E. van Nieuwenburg, A NEAT quantum error decoder, SciPost Phys. **11**, 005 (2021).

[79] K. Meinerz, C.-Y. Park, and S. Trebst, Scalable neural decoder for topological surface codes, Phys. Rev. Lett. **128**, 080505 (2022).

[80] R. W. 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).

[81] 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).

[82] M. Zhang, X. Ren, G. Xi, Z. Zhang, Q. Yu, F. Liu, H. Zhang, S. Zhang, and Y.-C. Zheng, A scalable, fast and programmable neural decoder for fault-tolerant quantum computation using surface codes, arXiv:2305.15767.

[83] T. Wagner, H. Kampermann, D. Bruß, and M. Kliesch, Pauli channels can be estimated from syndrome measurements in quantum error correction, Quantum **6**, 809 (2022).

[84] E. H. Chen, T. J. Yoder, Y. Kim, N. Sundaresan, S. Srinivasan, M. Li, A. D. Córcoles, A. W. Cross, and M. Takita, Calibrated decoders for experimental quantum error correction, Phys. Rev. Lett. **128**, 110504 (2022).

[85] T. Wagner, H. Kampermann, D. Bruß, and M. Kliesch, Learning logical pauli noise in quantum error correction, Phys. Rev. Lett. **130**, 200601 (2023).

[86] H. Bombin and M. A. Martin-Delgado, Optimal resources for topological two-dimensional stabilizer codes: Comparative study, Phys. Rev. A **76**, 012305 (2007).

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

[88] D. K. Tuckett, A. S. Darmawan, C. T. Chubb, S. Bravyi, S. D. Bartlett, and S. T. Flammia, Tailoring surface codes for highly biased noise, Phys. Rev. X **9**, 041031 (2019).

[89] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, arXiv:1609.02907.

[90] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Networks Learn. Syst. **32**, 4 (2021).

[91] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, Benchmarking graph neural networks, arXiv:2003.00982.

[92] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks, arXiv:1810.02244.

[93] M. Fey and J. E. Lenssen, Fast graph representation learning with pytorch geometric, arXiv:1903.02428.

[94] O. Higgott and C. Gidney, Pymatching v2, https://github.com/oscarhiggott/PyMatching (2022).

[95] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, Graph attention networks, arXiv:1710.10903.

[96] J. Lee, I. Lee, and J. Kang, Self-attention graph pooling, arXiv:1904.08082.

[97] B. Knyazev, G. W. Taylor, and M. R. Amer, Understanding attention and generalization in graph neural networks, arXiv:1905.02850.

[98] H. Gao and S. Ji, Graph u-nets, arXiv:1905.05178.

[99] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, Towards sparse hierarchical graph classifiers, arXiv:1811.01287.

[100] D. K. Tuckett, Tailoring surface codes: Improvements in quantum error correction with biased noise, Ph.D. thesis, University of Sydney (2020) (qecsim: https://github.com/qecsim/qecsim).

[101] B. Barber, K. M. Barnes, T. Bialas, O. Buğdayci, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric, C. Topal, M. L. Turner, and A. B. Ziad, A real-time, scalable, fast and resource-efficient decoder for a quantum computer, Nat. Electronics **8**, 84 (2025).

[102] D. K. Tuckett, S. D. Bartlett, and S. T. Flammia, Ultrahigh error threshold for surface codes with biased noise, Phys. Rev. Lett. **120**, 050505 (2018).

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

[104] A. Dua, A. Kubica, L. Jiang, S. T. Flammia, and M. J. Gullans, Clifford-deformed surface codes, PRX Quantum **5**, 010347 (2024).

[105] K. Tiurev, P.-J. H. S. Derks, J. Roffe, J. Eisert, and J.-M. Reiner, Correcting non-independent and non-identically distributed errors with surface codes, Quantum **7**, 1123 (2023).

[106] E. Huang, A. Pesah, C. T. Chubb, M. Vasmer, and A. Dua, Tailoring three-dimensional topological codes for biased noise, PRX Quantum **4**, 030338 (2023).

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

[108] H. Bombín, Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes, New J. Phys. **17**, 083002 (2015).

[109] J. R. Wootton, A family of stabilizer codes for $D(Z_2)$ anyons and majorana modes, J. Phys. A: Math. Theor. **48**, 215302 (2015).

[110] J. R. Wootton, Hexagonal matching codes with 2-body measurements, arXiv:2109.13308.

[111] B. Srivastava, A. Frisk Kockum, and M. Granath, The XYZ^2 hexagonal stabilizer code, Quantum **6**, 698 (2022).

[112] B. Hetényi and J. R. Wootton, Tailoring quantum error correction to spin qubits, Phys. Rev. A **109**, 032433 (2024).

[113] N. P. Breuckmann and J. N. Eberhardt, Quantum low-density parity-check codes, PRX Quantum **2**, 040101 (2021).

[114] P. Panteleev and G. Kalachev, Asymptotically good quantum and locally testable classical ldpc codes, arXiv:2111.03654.

[115] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, Nature (London) **627**, 778 (2024).

[116] J. Haah and M. B. Hastings, Boundaries for the Honeycomb Code, arXiv:2110.09545.

[117] M. S. Kesselring, J. C. M. de la Fuente, F. Thomsen, J. Eisert, S. D. Bartlett, and B. J. Brown, Anyon condensation and the color code, PRX Quantum **5**, 010342 (2024).

[118] A. deMarti iOlius, J. E. Martinez, P. Fuentes, P. M. Crespo, and J. Garcia-Frias, Performance of surface codes in realistic quantum hardware, Phys. Rev. A **106**, 062428 (2022).

[119] K. Tiurev, P.-J. H. S. Derks, J. Roffe, J. Eisert, and J.-M. Reiner, Correcting non-independent and non-identically distributed errors with surface codes, arXiv:2208.02191.

[120] C. A. Pattison, M. E. Beverland, M. P. da Silva, and N. Delfosse, Improved quantum error correction using soft information, arXiv:2107.13589.

[121] https://github.com/LangeMoritz/GNN_decoder.

[122] B. M. Varbanov, M. Serra-Peralta, D. Byfield, and B. M. Terhal, Neural network decoder for near-term surface-code experiments, Phys. Rev. Res. **7**, 013029 (2025).

[123] J. Bausch, A. W. Senior, F. J. Heras, T. Edlich, A. Davies, M. Newman, C. Jones, K. Satzinger, M. Y. Niu, S. Blackwell *et al.*, Learning high-accuracy error decoding for quantum processors, Nature (London) **635**, 834 (2024).