Neural Belief-Matching Decoding for Topological Quantum Error Correction Codes

Luca Menti, Francisco Lázaro
Institute of Communications and Navigation
German Aerospace Center (DLR)
email: {Luca.Menti, Francisco.LazaroBlasco}@dlr.de

***Abstract*—Quantum error correction (QEC) is critical for scalable fault-tolerant quantum computing. Topological codes, such as the toric code, offer hardware-efficient architectures but their Tanner graphs contain many girth-4 cycles that degrade the performance of belief-propagation (BP) decoding. For this reason, BP decoding is typically followed by a more complex second stage decoder such as minimum-weight perfect matching. These combined decoders achieve a remarkable performance, albeit at the cost of increased complexity. In this paper we propose two key improvements for the decoding of toric code. The first one is replacing the BP decoder by a neural BP decoder, giving rise to the neural belief-matching decoder which substantially decreases the average decoding complexity. The main drawback of this approach is the high cost associated with the training of the neural BP decoder. To address this issue, we impose a convolutional architecture on the neural BP decoder, enabling weight sharing across the spatially homogeneous structure of the code's factor graph. This design allows a model trained on a modest-size topological code to be directly transferred to much larger instances, preserving decoding quality while dramatically lowering the training burden. Our numerical experiments on toric-code lattices of various sizes demonstrate that this technique does not result in a noticeable loss in performance.**

***Index Terms*—quantum error correction, topological quantum error correcting codes, decoding algorithms, machine learning**

I. Introduction

Quantum error correction (QEC) schemes are essential to realizing fault-tolerant quantum computation. Since the early breakthroughs, the field has grown dramatically, and today we have multiple families of quantum error correction codes (QECC), including topological codes [1], Quantum Low-Density Parity Check (QLDPC) codes [2] and Quantum Turbo Codes (QTC) [3]. Topological codes offer several key advantages. They operate using only local, nearest-neighbor interactions and every stabilizer measurement involves only a few adjacent qubits. For these reasons, they are regarded as the preferred QEC solution for superconducting-qubit platforms [4], [5].

Among the many different decoding algorithms that have been proposed for topological codes, the most widely used is the minimum weight perfect matching (MWPM) decoder [6], [7] which provides an excellent error-correction capability, albeit at the cost of a large decoding complexity. In particular its worst-case runtime is $O(n^3 \log(n))$, with n being the number of (physical) qubits [6]–[8].

MWPM is optimal[1] under independent bit- and phase-flip errors. However, it is suboptimal under the depolarizing channel due to the presence of correlated $\boldsymbol{X}-\boldsymbol{Z}$ (i.e., $\boldsymbol{Y}$) errors.

A very appealing decoding algorithm is BP, which exhibits a low decoding complexity, is easily parallelized in hardware implementations and can inherently deal with $\boldsymbol{X}-\boldsymbol{Z}$ (i.e., $\boldsymbol{Y}$) errors. However, the performance of topological codes under BP decoding is very poor due to the large number of girth-4 cycles in their Tanner graph.

An effective decoding approach is using BP as a first stage decoding and feeding its output into a second stage MWPM decoder, giving rise to the belief-matching decoder [9]. This technique not only improves the error-correcting performance but also yields a lower average decoding complexity, since the second stage decoder only needs to be used when BP fails to converge.

Another interesting approach to improve the performance of BP decoding is neural BP [10], which reinterprets the decoder as a neural network and adds trainable multiplicative weights to the exchanged messages. This technique was originally introduced for classical error correcting codes but has also been proposed for QEC codes [11], [12]. The main drawback of this technique is that training becomes increasingly challenging (in terms of computational and memory resources) for larger code distances. This problem becomes even more acute when considering circuit-level noise. Moreover, its overall performance remains inferior to that of the belief-matching under the phenomenological depolarizing noise model, as it is shown in this paper.

The contribution of this paper is twofold. First, we propose replacing the initial BP stage in belief-matching by neural BP. As we show in our numerical results, this drastically reduces the average decoding complexity. In particular, the number of times that the MWPM decoder is called is reduced by up to 4 orders of magnitude. The second contribution is an approach to dramatically reduce the training cost of neural BP for topological codes. In particular, we exploit the translational symmetry of the two dimensional lattice and introduce a convolutional structure in the neural BP decoder. This approach enables to train our decoder on a small topological code, and reuse the

This work is part of the HESC lighthouse project within Munich Quantum Valley initiative and is supported by the Bavarian state government with funds from the Hightech Agenda Bavaria.

Luca Menti's research was funded by a scholarship awarded by the German Academic Exchange Service (DAAD).

[1] it returns the most likely error

obtained weights for larger codes. Our results indicate that this approach is effective and does not incur any noticeable loss in performance.

II. Background

A. Preliminaries

The n-qubit Pauli group (phases omitted) is $\boldsymbol{\mathcal{P}} = \{\boldsymbol{P}_1 \otimes \cdots \otimes \boldsymbol{P}_n \mid \boldsymbol{P}_i \in \{\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}\}\}$. The weight w of a Pauli operator $\boldsymbol{P} = \boldsymbol{P}_1 \otimes \cdots \otimes \boldsymbol{P}_n$ is the number of non-identity tensor factors. A $[[n, k, d]]$ stabilizer code is defined by an abelian subgroup $\boldsymbol{\mathcal{S}} \subset \boldsymbol{\mathcal{P}}$ generated by $m = n - k$ independent, commuting Pauli checks; it encodes k logical qubits into n physical qubits, has minimum distance d and can correct any error acting up to $\lfloor (d-1)/2 \rfloor$ qubits. The code space $\mathcal{C}$ is defined as the set of states $|\psi\rangle$ for which $\boldsymbol{S}|\psi\rangle = |\psi\rangle$ holds for all $\boldsymbol{S} \in \boldsymbol{\mathcal{S}}$ [13]–[15].
Writing a Pauli as $\boldsymbol{\mathcal{P}} = \bigotimes_{i=1}^{n} \boldsymbol{X}^{x_i} \boldsymbol{Z}^{z_i}$ maps it to the binary vector $(\mathbf{x} \mid \mathbf{z}) \in \mathbb{F}_2^{2n}$. Two Paulis commute iff their symplectic product vanishes, $\sum_{i=1}^{n} (x_i b_{Z,i} + z_i b_{X,i}) \equiv 0 \pmod 2$. Hence, a stabilizer code is described by a binary parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{(2n-k)\times 2n}$.
For Calderbank–Shor–Steane (CSS) codes $\boldsymbol{H}$ block-diagonalizes as $\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_X & 0 \\ 0 & \boldsymbol{H}_Z \end{pmatrix}$ [13].

The Pauli operators can also be represented as elements of the finite field $\mathbb{F}_4 = \{0, 1, \omega, \bar{\omega}\}$. Each Pauli string is encoded via the mapping $(x_i, z_i) \mapsto \alpha_i = x_i \omega + z_i \bar{\omega}$, where $(x_i, z_i) \in \{0,1\}^2$ denotes the binary $\boldsymbol{X}$- and $\boldsymbol{Z}$-type coordinates. This transformation converts the binary check matrix $\boldsymbol{H}$ into a quaternary stabilizer matrix $\boldsymbol{G} \in \mathbb{F}_4^{(2n-\log_2 n)\times n}$.

The normalizer of the stabilizer group $\mathcal{S}$, denoted $\mathcal{N}(\mathcal{S})$, consists of all Pauli operators commuting with every element of $\mathcal{S}$. $\mathcal{N}(\mathcal{S})$ is generated by $2n - m$ independent operators, where m is the number of stabilizer generators. The matrix $\boldsymbol{G}^{\perp} \in \mathbb{F}_4^{(2n-m)\times n}$ contains the generators of $\mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$ as rows, representing the logical operators. The code's minimum distance d equals the smallest weight of any such logical operator.

For an error $\boldsymbol{E} \in \boldsymbol{\mathcal{P}}$, measuring all stabilizer generators yields the syndrome $\boldsymbol{s} \in \{\pm 1\}^m$. Let $\hat{\boldsymbol{E}}$ be the error estimated provided by the decoder. Error correction is successful iff $\boldsymbol{E}\hat{\boldsymbol{E}} \in \boldsymbol{\mathcal{S}}$. In this work we adopt the phenomenological depolarizing noise model, in which each qubit independently undergoes a Pauli $\boldsymbol{X}$, $\boldsymbol{Y}$, or $\boldsymbol{Z}$ error with probability $\epsilon/3$, so that the overall physical error rate is ϵ.

B. Toric code

The toric code is a topological stabilizer code on a torus encoding two logical qubits [1]. A $\left[\left[n = 2d^2, k = 2, d\right]\right]$ toric code is obtained by placing qubits on the edges of a $d \times d$ square lattice with periodic boundary conditions. The stabilizer group is generated by weight-4 vertex and plaquette checks: vertex operators act as Pauli-$\boldsymbol{X}$ on the four edges incident to each vertex, and plaquette operators act as Pauli-$\boldsymbol{Z}$ on the four edges surrounding each face. A $d \times d$ toric code therefore has d^2 $\boldsymbol{X}$–type and d^2 $\boldsymbol{Z}$–type stabilizers, with one of each type redundant due to global constraints on the torus. A schematic of the toric code is shown in Fig. 1.

By multiplying adjacent vertex/plaquette operators, one obtains two weight-6 stabilizers for each vertex/plaquette operator. Together with the n independent weight-4 checks, these define the overcomplete $3n \times n$ check matrix used in this work[2].

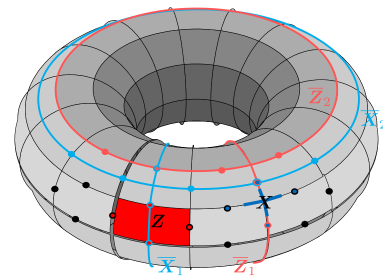


Fig. 1: Illustration of the toric code. Edge qubits (dots) are measured by weight-4 stabilizers $\boldsymbol{X}$-type vertex checks (blue) and $\boldsymbol{Z}$-type plaquette checks (red). Light-blue and light-red non-contractible strings denote the logical operators $\overline{\boldsymbol{X}}_{1,2}$ and $\overline{\boldsymbol{Z}}_{1,2}$ that wrap the two cycles of the torus.

C. Quaternary belief-propagation

We use the log-domain quaternary BP decoder of [16] on the Tanner graph defined by $\boldsymbol{G}$. The graph has variable nodes (VNs) $\{v_i \mid i = 1, \ldots, n\}$ (error symbols) and check nodes (CNs) $\{c_j \mid j = 1, \ldots, m\}$ (syndrome bits). An edge connects v_i and c_j iff $\boldsymbol{G}_{ij} \neq 0$, and the edge is labeled by $\boldsymbol{G}_{ij} \in \mathbb{F}_4$. Given a measured syndrome $\boldsymbol{s} \in \{0,1\}^m$, BP iteratively exchanges log-messages along the edges to approximate the posterior marginal distribution of the error symbols (variable nodes). Compared to using two separate binary BP decoders for $\boldsymbol{X}$ and $\boldsymbol{Z}$ checks, this quaternary decoder explicitly accounts for the correlation between $\boldsymbol{X}$ and $\boldsymbol{Z}$ errors induced by the depolarizing channel.
For each variable node v_i we initialize the log-likelihood ratio (LLR) vector $\boldsymbol{\Gamma}_{i\to j}$ as $\boldsymbol{\Lambda}_i = \left(\Lambda_i^{(1)}\ \Lambda_i^{(\omega)}\ \Lambda_i^{(\bar{\omega})}\right) \in \mathbb{R}^3$, with $\Lambda_i^{(\zeta)} = \ln\left(\frac{P(e_i=0)}{P(e_i=\zeta)}\right) = \ln\left(\frac{1-\epsilon}{\epsilon/3}\right)$. For each nonzero $\eta \in \boldsymbol{G}$, define $\phi_\eta : \mathbb{R}^3 \to \mathbb{R}$, mapping an LLR vector to the scalar LLR of the binary variable $\langle e_i, \eta\rangle$ (inner product over $\mathbb{F}_2$):

$$\phi_\eta(\boldsymbol{\Lambda}_i) = \ln\left(\frac{P(\langle e_i, \eta\rangle = 0)}{P(\langle e_i, \eta\rangle = 1)}\right) = \ln\left(\frac{1 + e^{-\Lambda_i^{(\eta)}}}{\sum_{\zeta\neq 0, \zeta\neq\eta} e^{-\Lambda_i^{(\zeta)}}}\right).$$

The initial scalar VN-to-CN messages are computed as $\phi_{i\to j} := \phi_{G_{j,i}}(\boldsymbol{\Gamma}_{i\to j})$ where $i \to j$ denotes the message from VN v_i to CN c_j. The outgoing messages from CN c_j are

$$\Delta_{i\leftarrow j} = (-1)^{s_j} \cdot 2\tanh^{-1}\left(\prod_{i'\in\mathcal{N}(j)\setminus\{i\}} \tanh\frac{\phi_{i'\to j}}{2}\right), \quad (1)$$

[2]As shown in [12], augmenting the original full-rank parity-check matrix with such redundant rows significantly improves BP and quaternary neural belief-propagation decoding performance on toric codes.

where $\mathcal{N}(j)$ denotes the neighboring VNs of c_j.
In the VN update, we compute $\mathbf{\Gamma}_{i\to j} = \left(\Gamma^{(1)}_{i\to j}\ \Gamma^{(\omega)}_{i\to j}\ \Gamma^{(\bar{\omega})}_{i\to j}\right)$ with

$$\Gamma^{(\zeta)}_{i\to j} = \Lambda^{(\zeta)}_i + \sum_{\substack{j'\in\mathcal{M}(i)\setminus\{j\}\\ \langle\zeta,\, G_{j',i}\rangle=1}} \Delta_{i\leftarrow j'}, \tag{2}$$

for all $\zeta \in \mathbb{F}_4\setminus\{0\}$, where $\mathcal{M}(i)$ denotes the neighboring CNs of v_i. The outgoing VN-to-CN messages are then $\phi_{i\to j} = \phi_{G_{j,i}}\left(\mathbf{\Gamma}_{i\to j}\right)$.
To form the final decision, we compute for each $i \in \{1, \ldots, n\}$

$$\Gamma^{(\zeta)}_i = \Lambda^{(\zeta)}_i + \sum_{\substack{j\in\mathcal{M}(i)\\ \langle\zeta,\, G_{j,i}\rangle=1}} \Delta_{i\leftarrow j}, \tag{3}$$

for all $\zeta \in \mathbb{F}_4 \setminus \{0\}$, and decide $\hat{e}_i = \arg\min_\zeta \Gamma^{(\zeta)}_i$. The iterations proceed until the syndrome is satisfied or until a maximum predefined number of iterations is reached.

D. Quaternary neural belief-propagation

We use the quaternary neural belief-propagation (NBP) of [12], which introduces the trainable weights on the edges of the Tanner graph, giving rise to the following message update rules

$$\Delta_{i\leftarrow j} = (-1)^{s_j}\cdot 2\tanh^{-1}\left(\prod_{i'\in\mathcal{N}(j)\setminus\{i\}} \tanh\frac{w^{(l)}_{v,i',j}\cdot\phi_{i'\to j}}{2}\right), \tag{4}$$

and

$$\Gamma^{(\zeta)}_{i\to j} \;=\; w^{(l)}_{v,i}\Lambda^{(\zeta)}_i \;+ \sum_{\substack{j'\in\mathcal{M}(i)\setminus\{j\}\\ \langle\zeta,\, G_{j',i}\rangle=1}} w^{(l)}_{c,i,j'}\cdot\Delta_{i\leftarrow j'}, \tag{5}$$

Here, l denotes the iteration index and $w^{(l)}_{v,i',j}$, $w^{(l)}_{c,i,j'}$, and $w^{(l)}_{v,i}$ are the real-valued trainable weights associated to the VN-to-CN messages, CN-to-VN messages, and channel log-likelihood vectors, respectively.
For training, we adopt the loss function of [12] that takes into account degeneracy:

$$\mathcal{L}(\mathbf{\Gamma}; e) = \sum_{j=1}^{2n-m} f\left(\sum_{i=1}^{n} P\left(\langle e_i + \hat{e}_i, G^{\perp}_{j,i}\rangle = 1 \mid \boldsymbol{s}\right)\right) \tag{6}$$

where $P\left(\langle\hat{e}_i, \eta\rangle = 1 \mid \boldsymbol{s}\right) = \left(1 + \mathrm{e}^{-\phi_\eta(\mathbf{\Gamma}_i)}\right)^{-1}$, and $f(x) = |\sin(\pi x/2)|$ [11]. The loss $\mathcal{L}$ is minimized when $\langle \boldsymbol{e} + \hat{\boldsymbol{e}},\ \boldsymbol{G}^{\perp}_i\rangle = 0$ holds for every row $i \in \{1, 2, \ldots, 2n-m\}$ of $\boldsymbol{G}^{\perp}$.

E. Belief-matching

The Belief-matching decoder couples binary BP with MWPM [9], [17] [3]. In the first stage, BP is run on the Tanner graph yielding posterior marginals for each error mechanism (variable node). If these define a valid correction, it is accepted. Otherwise, the posteriors are used to *reweight* the matching graph by decomposing each hyperedge into edges and redistributing probability mass, which is then converted into edge weights via a logarithmic map. Finally, weighted MWPM is applied to obtain a syndrome-consistent correction. Unlike standard matching decoders, which use only prior error rates and ignore hyperedge correlations, this scheme injects posterior information into the matching step, allowing BP to capture correlations such as those from $\boldsymbol{Y}$ errors and improving practical decoding performance.

III. Proposed Decoders

A. Neural belief-matching

We propose a novel two-stage decoder, quaternary neural belief-matching (NBP-matching), that combines a neural belief-propagation pre-processor (NBP) with minimum-weight perfect matching (MWPM). The method builds on the belief-matching algorithm of [9] by replacing the binary BP step with NBP [4].

In particular, the decoder workflow is as follows. We first run NBP, which provides a log-likelihood-ratio vector $\mathbf{\Gamma}_i$ for each variable node v_i. If NBP converges to a valid solution we terminate. Alternatively, i.e., if the output of NBP does not match the observed syndrome $\boldsymbol{s}$, the output of NBP is fed to the MWPM algorithm to compute the final correction.

The key advantage of using NBP over traditional BP lies in its ability to better handle degeneracy and loops in the Tanner graph, which are critical challenges in decoding topological codes. As a consequence, MWPM needs to be called much less frequently, which yields a reduction of the average computational complexity.

B. Convolutional neural belief-propagation and weight-reuse strategy

The quaternary convolutional neural belief-propagation (convolutional NBP) decoder introduced in this work is a convolutional neural networks (CNN)-inspired architecture that leverages the toric code's periodic boundary conditions to implement weight-sharing, a fundamental feature of convolutional neural networks [18]. This approach is inspired by the weight-sharing framework introduced for toric codes in [11], and by works on classical error correction [19], [20].

The defining feature of CNN is the convolutional layer, with the convolutional operation defined as: $o_{i,j} = h\left(\sum_{q=1}^{D}(\mathbf{K}_q * \mathbf{X}_q)(i,j) + b\right)$ where $\mathbf{K}$ is the kernel, $\mathbf{X}$ is the input tensor, $o_{i,j}$ is the output of the layer (feature map) associated with the weights $w_{i,j}$, h is the activation function, b is the bias and D the number of filters [21].
To adapt this operation to our toric code, we define three kernel matrices: $\boldsymbol{G}^{\text{VN}}_{\text{type}}$ and $\boldsymbol{G}^{\text{CN}}_{\text{type}}$ for weights associated to variable and check node messages (size $m \times n$) and $\boldsymbol{G}^{\text{channel}}_{\text{type}}$ for weights associated to the channel log-likelihood vectors (size $1 \times n$). The kernel matrices are obtained by tiling a

[3]Belief-matching, as originally introduced in [9], relies on binary BP. However, in this work we adopt quaternary BP.

[4]Implementation taken from [12] (https://github.com/kit-cel/Quantum-Neural-BP4-demo).

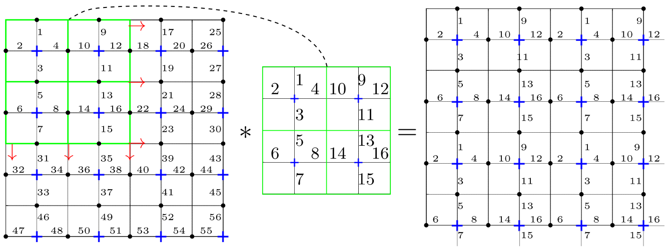


Fig. 2: Construction of $\boldsymbol{G}_{\text{type}}^{\text{CN}}$ for $\boldsymbol{X}$-stabilizers on a $d = 4$ toric code. A 2×2 patch of $\boldsymbol{X}$-stabilizers (and their edges) is tiled over the lattice, enforcing weight-sharing and translation invariance; the $\boldsymbol{Z}$-stabilizers are built analogously.

$$\overset{\boldsymbol{G}_{\text{type}}^{\text{CN}}}{\begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 5 & 6 & 0 \\ 0 & 0 & 1 & 2 \\ 6 & 0 & 0 & 5 \end{pmatrix}} * \overset{\boldsymbol{w}_{CN}}{\begin{pmatrix} w_{\mathrm{c},1,1}^{(l)} & w_{\mathrm{c},1,2}^{(l)} & w_{\mathrm{c},1,3}^{(l)} & w_{\mathrm{c},1,4}^{(l)} \\ w_{\mathrm{c},2,1}^{(l)} & w_{\mathrm{c},2,2}^{(l)} & w_{\mathrm{c},2,3}^{(l)} & w_{\mathrm{c},2,4}^{(l)} \\ w_{\mathrm{c},3,1}^{(l)} & w_{\mathrm{c},3,2}^{(l)} & w_{\mathrm{c},3,3}^{(l)} & w_{\mathrm{c},3,4}^{(l)} \\ w_{\mathrm{c},4,1}^{(l)} & w_{\mathrm{c},4,2}^{(l)} & w_{\mathrm{c},4,3}^{(l)} & w_{\mathrm{c},4,4}^{(l)} \end{pmatrix}} = \overset{\boldsymbol{w}_{CN}^{*}}{\begin{pmatrix} \frac{w_{\mathrm{c},1,1}^{(l)}+w_{\mathrm{c},3,3}^{(l)}}{2} & \frac{w_{\mathrm{c},1,2}^{(l)}+w_{\mathrm{c},3,4}^{(l)}}{2} & 0 & 0 \\ 0 & \frac{w_{\mathrm{c},2,2}^{(l)}+w_{\mathrm{c},4,4}^{(l)}}{2} & \frac{w_{\mathrm{c},2,3}^{(l)}+w_{\mathrm{c},4,1}^{(l)}}{2} & 0 \\ 0 & 0 & \frac{w_{\mathrm{c},1,1}^{(l)}+w_{\mathrm{c},3,3}^{(l)}}{2} & \frac{w_{\mathrm{c},1,2}^{(l)}+w_{\mathrm{c},3,4}^{(l)}}{2} \\ \frac{w_{\mathrm{c},2,3}^{(l)}+w_{\mathrm{c},4,1}^{(l)}}{2} & 0 & 0 & \frac{w_{\mathrm{c},2,2}^{(l)}+w_{\mathrm{c},4,4}^{(l)}}{2} \end{pmatrix}}$$

Fig. 3: Convolution in the convolutional NBP decoder: the check-node weight tensor $\boldsymbol{W}_{CN}$ is filtered by $\boldsymbol{G}_{\text{type}}^{\text{CN}}$ to enforce weight-sharing, and average-pooling over identical pivots produces the final tensor $\boldsymbol{W}_{CN}^{*}$.

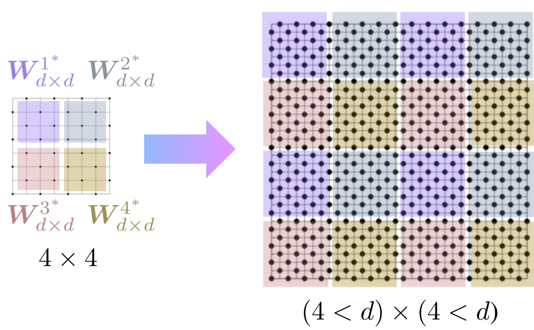


Fig. 4: Weight-reuse across code distances: the tensors $\boldsymbol{W}_{d\times d}^{n*}$ for check nodes, variable nodes, and LLRs trained on a $d = 4$ toric code are tiled to construct decoders for larger d, eliminating the need for costly retraining.

2×2 stabilizer patch across the toric lattice (Fig. 2). The patch contains a fixed set of stabilizers, and all instances of the same stabilizer type share the same weight index.

Applying a kernel to the weight tensor of the code reproduces the CNN behaviour: the same filter is applied over the entire lattice (stride 2, padding realized by the torus's periodic boundaries). The resulting activation map o is passed to the next layer, and the bias b is set to zero. Average pooling over entries that share the same kernel index yields the weight-shared tensors $\boldsymbol{W}_{\text{CN}}^{*}$, $\boldsymbol{W}_{\text{VN}}^{*}$, and $\boldsymbol{W}_{\text{channel}}^{*}$ (Fig. 3). The final fully-connected layer is the NBP network described in Sec. II-D.

Weight-sharing drastically reduces the number of trainable parameters, leading to faster training, lower memory usage, and enabling deeper networks [21]–[23]. Because the decoder is built from a 2×2 patch, the weights learned for a code of distance d can be tiled to decode any larger (or smaller) toric lattice with even dimension L without retraining (Fig. 4).

IV. Numerical Results

In this section we compare the performance of the proposed two-stage decoding algorithms to existing algorithms. We consider toric codes of distances $4, 6, 8$ and 10, and physical qubit error rates ranging from 0.015 to 0.15. The results are obtained through Monte Carlo simulations.

The figures in this section show 0.975 confidence intervals[5] for each data point, computed using a negative binomial estimator [24].

A. NBP-Matching Results

Here, we evaluate the performance of belief-matching, NBP-matching, and RNBP-matching (a variant of NBP with overcomplete check matrix (RNBP)) against the RNBP decoder proposed in [12] and the MWPM decoder. The NBP's training methodology employed in this work is the one proposed in [12]. The MWPM results are generated using the Pymatching library [8].

Fig. 5 shows the logical error rate (LER) as a function of the physical qubit error rate for the aforementioned decoders. We can observe that all three two-step decoders (belief-matching, NBP-matching and RNBP-matching) perform similarly. In particular, they outperform standalone MWPM by approximately one order of magnitude. For code distances $d = 4, 6$ RNBP achieves a similar performance to that of the two-stage decoders. However, for larger distances $d = 8$ and $d = 10$, the two-step decoders outperform RNBP.

Fig. 6 shows the frequency of MWPM algorithm invocations as a postprocessing step for the different two-stage decoders as a function of the physical qubit error rate. The results show that the standard neural BP decoder (without using overcomplete matrices), NBP-matching, reduces considerably the number of calls to MWPM. The number of calls is further reduced when using the RNBP-matching decoder. Thus, the use of overcomplete check matrices in the two-step decoder yields a significant reduction of the average computational cost of the decoder.

B. Convolutional NBP and weight-reuse strategy results

We first evaluate the performance of the proposed convolutional NBP with overcomplete check matrix (convolutional RNBP). That is, we consider for the moment only the first-stage of the proposed decoders, without matching. Fig. 7 shows the LER as a function of the physical qubit error rate for convolutional RNBP and RNBP, which is added for reference. The results demonstrate that CNN decoder performs similarly to neural networks (NN)-based one, despite having much fewer trainable weights. We now evaluate the performance of the proposed weight-reuse strategy. Fig. 8(a) shows the LER as a function of the physical qubit error rate for different decoders for a toric code of distance $d = 10$, including a convolutional RNBP decoder that reuses weights trained on a $d = 4$ toric code. As it can be observed, reusing the weights of $d = 4$ yields essentially the same performance compared to using weights trained for $d = 10$.

[5] the confidence intervals in several figures are too small to be visible at the plot scale, rendering them indistinguishable from the curves. The raw data and detailed statistical analyses are available upon request.

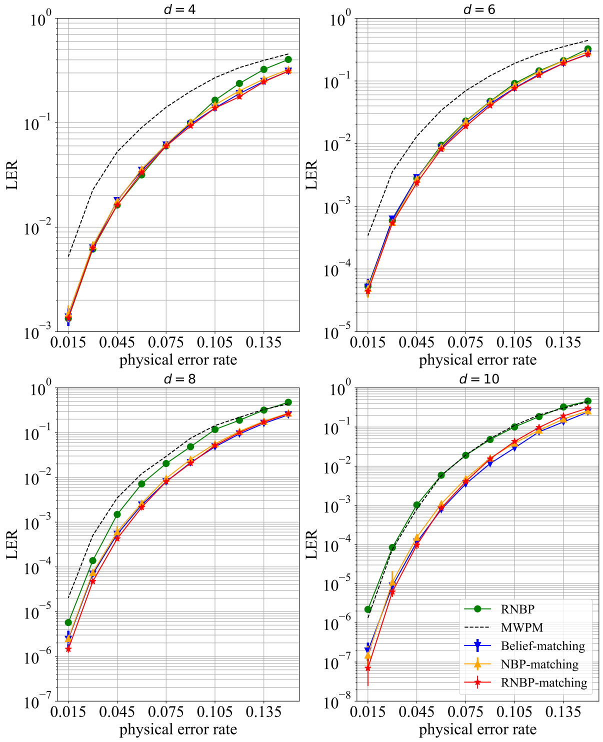


Fig. 5: LER versus physical error rate for toric codes ($d = 4, 6, 8, 10$) using belief-matching, NBP-matching, and RNBP-matching decoders. RNBP results are taken from [12].

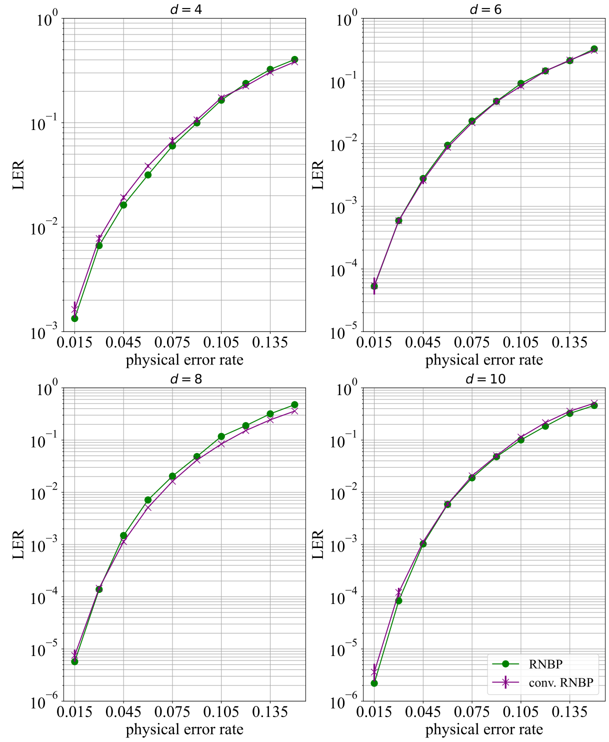


Fig. 7: LER versus physical error rate for toric codes ($d = 4, 6, 8, 10$) using the convolutional RNBP decoder.

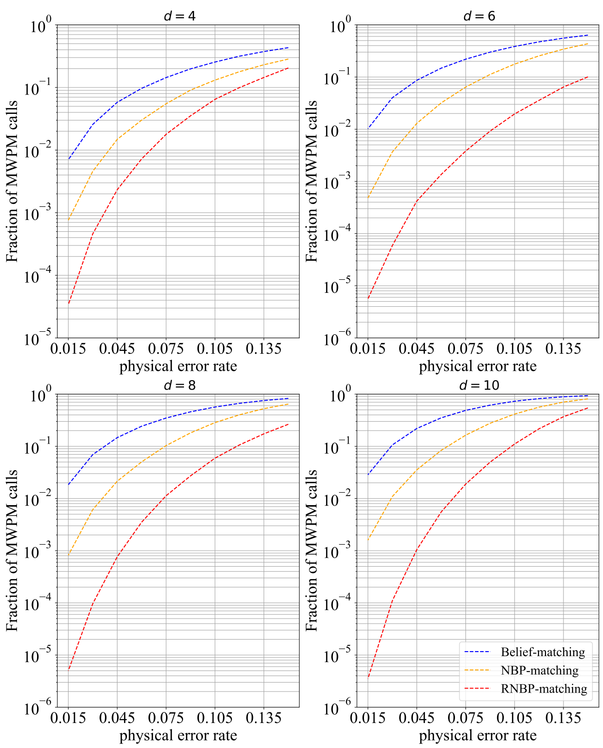


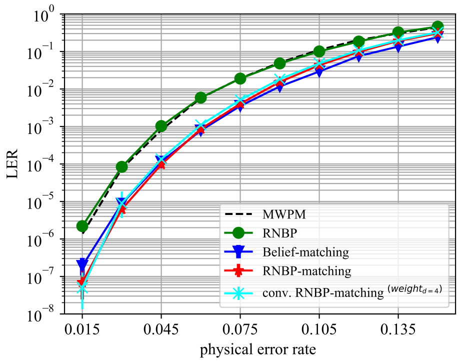
Fig. 6: Comparison of the frequency of MWPM algorithm calls as a post-processing step, as a function of the physical error rate, with dimensions $d \in \{4, 6, 8, 10\}$ utilizing the belief-matching, NBP-matching and RNBP-matching decoders.

Fig. 8(b) shows the frequency with which MWPM is called for belief-matching, RNBP-matching, and convolutional RNBP-matching reusing the weights obtained from a $d = 4$ toric code. As it can be observed, reusing the weights obtained for $d = 4$ for a code with $d = 10$, does not lead to more calls to MWPM.

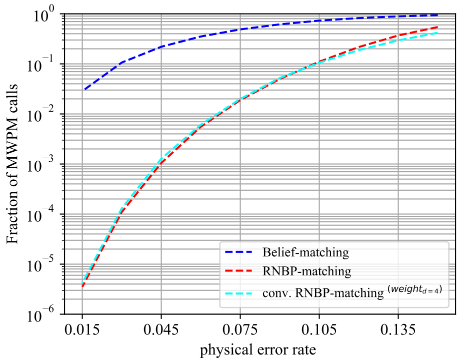
Thus, our results indicate that the weight reuse strategy introduced in Section III-B does not result in loss in performance. This is quite remarkable, since it solves the scalability problem of training neural networks of increasing size.

V. Conclusions

This paper has presented two significant advancements in the decoding of toric codes for quantum error correction. First, we introduced the neural belief-matching decoder, which replaces the traditional belief-propagation stage with a neural belief-propagation (NBP) decoder. This modification yields a dramatic reduction in the average decoding complexity, decreasing the number of calls to the minimum-weight perfect matching (MWPM) decoder by up to four orders of magnitude for larger code distances (e.g., $10^4\times$ fewer calls for $d = 8, 10$). Despite this substantial reduction in complexity, the neural belief-matching decoder maintains the same logical error rate performance compared to standard belief-matching. Second, we proposed a convolutional neural belief-propagation (convolutional NBP) decoder with a weight-reuse strategy that exploits the translational symmetry of the toric code. This approach enables training on a small code instance (e.g., $d = 4$) and direct application to much larger codes (e.g.,



(a)



(b)

Fig. 8: (a) LER comparison for toric codes using convolutional RNBP and convolutional RNBP-matching decoders, with weights trained on $d = 4$ applied to $d = 10$, and RNBP results from [12].
(b) Frequency of MWPM postprocessing invocations for convolutional RNBP-matching decoders under the same conditions.

$d = 10$) without any performance degradation. The weight-reuse strategy effectively addresses the scalability challenge of training neural decoders for larger quantum error correction codes, reducing the training burden while maintaining decoding quality. These contributions are particularly significant in the context of fault-tolerant quantum computing, where real-time decoding is essential. The exponential slowdowns highlighted by Terhal's backlog argument [5] become increasingly problematic as quantum systems scale, making the reduction in reliance on computationally intensive MWPM decoders critical for practical implementation. By minimizing the need for MWPM, our approach ensures that decoding can keep pace with syndrome generation, enabling scalable quantum error correction systems. The proposed techniques are not limited to toric codes and can be extended to other topological codes such as the surface code. The weight-reuse strategy shows particular promise for circuit-level noise models, where the Tanner graph complexity and training challenges are most pronounced. Future work will focus on extending these methods to more complex quantum error correction codes and exploring their application in practical quantum computing architectures.

VI. Acknowledgment

We thank A. Fengler and S. Miao for useful discussions.

References

[1] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[2] D. J. MacKay, G. Mitchison, and P. L. McFadden, "Sparse-graph codes for quantum error correction," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2315–2330, 2004.

[3] D. Poulin, J.-P. Tillich, and H. Ollivier, "Quantum serial turbo codes," *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2776–2798, 2009.

[4] A. G. Fowler, A. C. Whiteside, and L. C. Hollenberg, "Towards practical classical processing for the surface code," *Physical review letters*, vol. 108, no. 18, p. 180501, 2012.

[5] B. M. Terhal, "Quantum error correction for quantum memories," *Reviews of Modern Physics*, vol. 87, no. 2, pp. 307–346, 2015.

[6] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.

[7] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, vol. 17, pp. 449–467, 1965.

[8] O. Higgott, "Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–16, 2022.

[9] 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," *Physical Review X*, vol. 13, no. 3, p. 031007, 2023.

[10] E. Nachmani, Y. Be'Ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 341–346.

[11] Y.-H. Liu and D. Poulin, "Neural belief-propagation decoders for quantum error-correcting codes," *Physical Review Letters*, vol. 122, no. 20, p. 200501, 2019.

[12] S. Miao, A. Schnerring, H. Li, and L. Schmalen, "Quaternary neural belief propagation decoding of quantum ldpc codes with overcomplete check matrices," *IEEE Access*, 2025.

[13] D. Gottesman, "Stabilizer codes and quantum error correction phd thesis california institute of technology," *Preprint*, 1997.

[14] P. Fuentes, J. E. Martinez, P. M. Crespo, and J. Garcia-Frías, "Degeneracy and its impact on the decoding of sparse quantum codes," *IEEE Access*, vol. 9, pp. 89 093–89 119, 2021.

[15] T. Brun, I. Devetak, and M.-H. Hsieh, "Correcting quantum errors with entanglement," *science*, vol. 314, no. 5798, pp. 436–439, 2006.

[16] D. Ostrev, D. Orsucci, F. Lázaro, and B. Matuz, "Classical product code constructions for quantum calderbank-shor-steane codes," *Quantum*, vol. 8, p. 1420, 2024.

[17] Google Quantum AI, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.

[18] N. Ketkar and J. Moolayil, "Convolutional neural networks," in *Deep learning with Python: learn best practices of deep learning models with PyTorch*. Springer, 2021, pp. 197–242.

[19] L. Wang, C. Terrill, D. Divsalar, and R. D. Wesel, "Ldpc decoding with degree-specific neural message weights and rcq decoding," *IEEE Transactions on Communications*, vol. 72, no. 4, pp. 1912–1924, 2023.

[20] X. Chen and M. Ye, "Cyclically equivariant neural decoders for cyclic codes," *in Proc. Int. Conf. Mach. Learn. (ICML)*, pp. 1771–1780, 2021.

[21] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 2007.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 2002.

[24] B. Mazzeo and M. Rice, "On monte carlo simulation of the bit error rate," in *2011 IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–5.