# QUITS: A modular Qldpc code circUIT Simulator

Mingyu Kang[1,2], Yingjia Lin[1,2], Hanwen Yao[1,3], Mert Gökduman[1,3], Arianna Meinking[1,2], and Kenneth R. Brown[1,2,3,4]

[1]Duke Quantum Center, Duke University, Durham, NC 27701, USA
[2]Department of Physics, Duke University, Durham, NC 27708, USA
[3]Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA
[4]Department of Chemistry, Duke University, Durham, NC 27708, USA

arXiv:2504.02673v2 [quant-ph] 1 Dec 2025

**To achieve quantum fault tolerance with lower overhead, quantum low-density parity-check (QLDPC) codes have emerged as a promising alternative to topological codes such as the surface code, offering higher code rates. To support their study, an end-to-end framework for simulating QLDPC codes at the circuit level is needed. In this work, we present `QUITS`, a modular and flexible circuit-level simulator for QLDPC codes. Its design allows users to freely combine LDPC code constructions, syndrome extraction circuits, decoding algorithms, and noise models, enabling comprehensive and customizable studies of the performance of QLDPC codes under circuit-level noise. `QUITS` supports several leading QLDPC families, including hypergraph product codes, lifted product codes, and balanced product codes. As part of the framework, we introduce a syndrome extraction circuit improved from Tremblay, Delfosse, and Beverland [Phys. Rev. Lett. 129, 050504 (2022)] that applies to all three code families. In particular, for a small hypergraph product code, our circuit achieves lower depth than the conventional method, resulting in improved logical performance. Using `QUITS`, we evaluate the performance of state-of-the-art QLDPC codes and decoders under various settings, revealing trade-offs between the decoding runtime and the logical failure rate. The source code of `QUITS` is available online [1].**

Mingyu Kang: mingyu.kang@duke.edu, These authors contributed equally.

Yingjia Lin: yingjia.lin@duke.edu, These authors contributed equally.

## 1 Introduction

Quantum error correction (QEC) is a key requirement for building a scalable and fault-tolerant quantum computer. The theory of quantum fault tolerance establishes that the encoded quantum information can be protected even in the presence of *circuit-level noise*, a noise model in which errors may occur at any qubit, gate, and measurement involved in the QEC [1, 20].

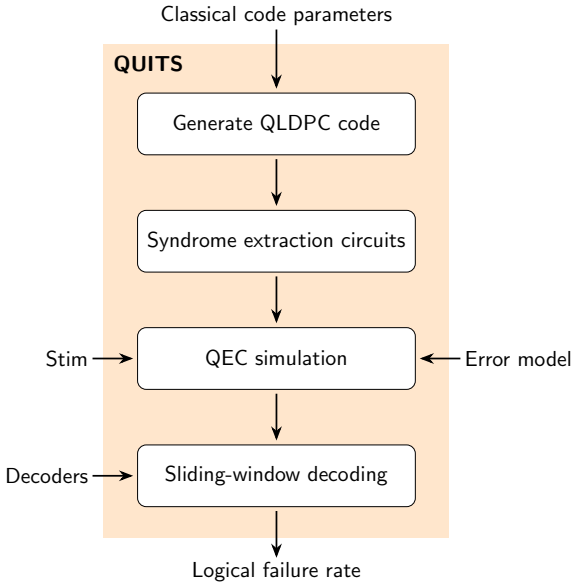



Figure 1: **Overview of `QUITS`.** QUITS provides a streamlined framework for QLDPC codes simulation. The shaded areas are the modularized functions of QUITS. Given code parameters and an error model as the input, QUITS calls Stim and customized decoders to perform QLDPC simulation and estimates the logical failure rate.

[1] `https://github.com/mkangquantum/quits`

The QEC performance heavily depends on factors such as the choice of the QEC code, the design of the circuit, and the implementation of the decoder. As such, a unified and modular framework is highly desired for evaluating the performance of each component.

Recently, research in QEC has been facilitated by software packages such as the stabilizer circuit simulator Stim [17] and the minimum-weight perfect matching decoder PyMatching [23]. For surface codes [16] in particular, Stim and PyMatching together offer a versatile end-to-end framework for simulating the circuit-level performance of surface codes [9, 24, 25, 43, 45, 51]. For higher rate codes such as quantum low-density parity-check (QLDPC) codes with non-local connectivity [8], circuit-level simulations have been performed in various works [6, 50, 52] using Stim; however, open-source frameworks that enable end-to-end simulations of the circuit are not widely developed, with the notable exception of Ref. [18] specifically for bivariate bicycle codes [6].

Even when the QLDPC code and the decoder are given, several challenges in performing circuit-level simulations remain. First, finding a circuit for extracting the syndromes of a QEC code in parallel is not straightforward from the code's parity-check matrices. Previous works often rely on a brute-force search to find the optimal scheduling of the entangling gates in the syndrome extraction circuit [3, 6, 35]. Second, decoders for QLDPC codes are often designed for the code-capacity noise model, where errors in gates and measurements are ignored. Additional work needs to be done to use the decoders for handling the correlated errors at the circuit level.

This work presents `QUITS`, a modular circuit simulator for QLDPC codes. Fig. 1 provides an overview of our software package and how it can be interfaced with other existing packages such as Stim [17] and LDPC decoders [41, 42]. Our simulator addresses the challenges in designing syndrome extraction circuits and implementing decoders for simulating QLDPC codes at the circuit level.

First, we provide a framework for writing the parallelized syndrome extraction circuit of QLDPC codes obtained from (generalized) products of two classical codes. Specifically, we improve the construction of the cardinal circuit, originally proposed for hypergraph product codes [50], and generalize to quasi-cyclic lifted product codes [33] and balanced product cyclic codes [47]. We expect our simulator to be helpful in finding the syndrome extraction circuit for newly developed QLDPC codes with a similar structure.

Second, we provide a modular framework for implementing decoders at the circuit level. In this framework, syndromes are divided into overlapping windows and passed to an *inner decoder* along with circuit-level noise information obtained from Stim [17], which then suggests a correction for each decoding window. The modular structure of this framework enables easy replacement of inner decoders, while the flexibility to adjust decoding window sizes mitigates the reduction in effective distance caused by the finite decoding window. Additionally, the conversion function from the Stim detector error model to window slices of the detector error matrix facilitates the analysis of various circuit-level noise models. As a result, this framework streamlines the development and testing of different decoders under circuit-level noise conditions.

This paper is organized as follows. In Sec. 2, we review some basics of QEC and QLDPC codes. In Sec. 3, we introduce the QLDPC codes considered in this work. We review the syndrome extraction circuit in Ref. [50] and discuss our improvements in Sec. 4. We introduce our decoding framework with a focus on its modular structure in Sec. 5. Finally, we present our numerical results in Sec. 6 and conclude the paper with outlooks in Sec. 7.

## 2 Background

Stabilizer codes of $n$ physical qubits encode $k$ logical qubits in the $+1$ eigenspace of an Abelian group $S$ of Pauli operators, the stabilizer group. The set of logical Pauli operators is all Pauli operators that commute with $S$ but are not in $S$. The weight of a Pauli operator is defined as the number of physical qubits it acts on non-trivially. The distance $d$ of a stabilizer code is the minimum weight of a logical Pauli operator. We denote a stabilizer code by the set of code parameters $[[n, k, d]]$.

A set of generators of $S$, or *check* operators, is measured at each cycle of QEC, where each measurement result is a *syndrome*. The minimum set of generators contains $n-k$ independent gen-

erators. Errors that bring the state outside the codespace flip some of the check operators and fire the corresponding syndromes. The *decoder* takes the syndromes as input and returns a suggested *correction* that reproduces the same syndrome. Applying this correction maps the state back to the codespace. If the correction and the error form a stabilizer, the code state is restored. Otherwise, they form a logical operator and create a logical error. Typically, a minimum-weight decoder is used, which chooses the correction with the lowest possible weight, or the error combination that occurs with the highest probability, that yields the same syndrome [13]. With a perfect minimum-weight decoder, up to $\lfloor (d-1)/2 \rfloor$ errors can be reliably corrected.

Calderbank-Shor-Steane (CSS) codes [10, 46] are stabilizer codes with a generator set containing either only Pauli-$Z$ operators or only Pauli-$X$ operators. We can represent the $P$-type ($P = Z, X$) checks using the parity check matrix $H_P$. The elements of the matrix are given by $H_P^{ij} = 1$ only if the $i$th $P$-type check has support on the $j$th qubit; otherwise, $H_P^{ij} = 0$. The check matrices satisfy the relation $H_X H_Z^T = 0$ such that all the checks commute. Measuring $Z(X)$-type checks can detect $X(Z)$-type errors that anticommute with the check operator. Thus, the $X$ and $Z$-type errors can be decoded independently.

The *Tanner graph*, a bipartite graph that represents the parity-check matrix of a classical or quantum code, is constructed as follows. One set of vertices represents the bits or qubits that encode the data, and the other set of vertices represents the checks. Edges connect each check vertex to each vertex that represents the bit or qubit that supports the check.

The circuit implementation of QEC involves *data qubits*, which contain the encoded information, and *syndrome qubits*, which are measured to reveal the syndromes. At each QEC cycle, each check operator is transferred to the corresponding syndrome qubit via *syndrome extraction circuit*, which includes multiple layers of two-qubit gates. The ordering of the two-qubit gates needs to be carefully designed to correctly extract the syndromes in parallel. QEC should be capable of protecting quantum information under the *circuit-level noise model*, where errors may occur at any qubit, gate, and measurement during the syndrome extraction circuit.

QLDPC codes are promising candidates for the implementation of QEC [8]. The low-density parity-check (LDPC) constraint states that for a given family of codes, the number of syndrome (data) qubits connected to each data (syndrome) qubit is bound by a constant independent of the code size. This allows the depth of the syndrome extraction circuit to be constant as the code size scales up, which is crucial for realistic implementation.

The most widely studied example is the surface code [15, 16, 37], where each data (syndrome) qubit is connected to its nearest-neighbor syndrome (data) qubits in a 2-dimensional lattice. Surface codes have several advantages, such as a high circuit-level error threshold and a planar layout of qubits, leading to the recent experimental realization of QEC below the threshold [19]. The drawback of surface codes is that they have asymptotically vanishing rates: as the lattice size $L \to \infty$, both the encoding rate $k/n = \Theta(L^{-2})$ and the distance rate $d/n = \Theta(L^{-1})$ vanish. This saturates the fundamental bounds on the parameters of QEC codes with 2-dimensional local connectivity [5].

By removing the constraint of local connectivity, various families of QLDPC codes with high encoding rate $k/n$ and distance rate $d/n$ are found [7, 29, 33, 34, 48]. The QEC performance of these codes depends on the choice of syndrome extraction circuit and the decoder implementation, which are more complicated than those for surface codes. The goal of this paper is to provide a modular framework for constructing the circuits and evaluating the decoders for the QLDPC codes with non-local connectivity.

## 3 QLDPC codes

We focus on QLDPC codes constructed from a product of two classical codes. The product structure admits a unified method for constructing the syndrome extraction circuit, as will be explained in Sec. 4.

### 3.1 Hypergraph product (HGP) codes

HGP codes [48] are the most straightforward example of QLDPC codes with product structure. Consider two classical LDPC codes with parity check matrix $H_\alpha$ and parameters $[n_\alpha, k_\alpha, d_\alpha]$

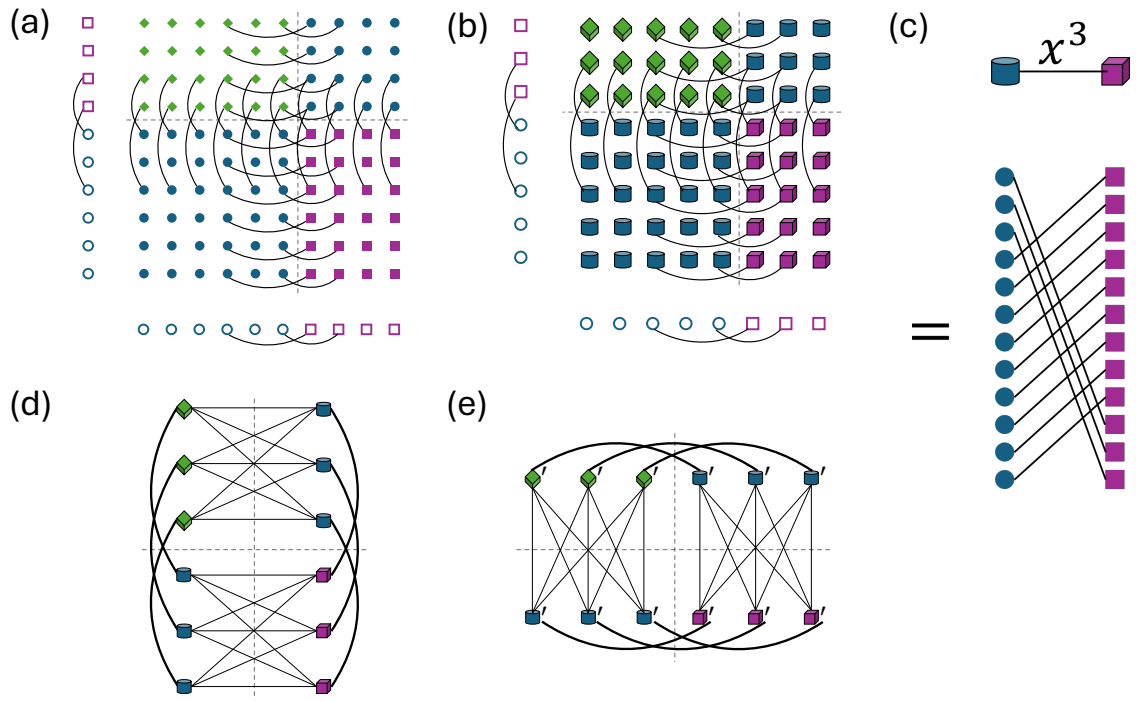



Figure 2: **QLDPC codes**. Filled blue circles, green diamonds, and purple squares represent the data, $Z$-type syndrome, and $X$-type syndrome qubits, respectively. 3-dimensional symbols represent the lifted nodes, each of which contains $l$ qubits of the corresponding type, where $l$ is the lift size. **(a)** Tanner graph of a hypergraph product code, constructed from a product of two classical codes. Empty blue circles and purple squares represent the data and check bits of the classical codes, respectively. **(b)** Tanner graph of a quasi-cyclic lifted product code. Each edge is assigned a monomial $x^s$ ($s \in [l]$, $x^l = 1$). **(c)** Example of unraveling the edge between two lifted nodes ($l = 12$). **(d)(e)** Tanner graph of a balanced product cyclic code. Each edge is assigned a polynomial with terms $x^s$. In (d), vertical edges have a one-to-one mapping. In (e), the qubits are "shuffled" within each quadrant such that horizontal edges have a one-to-one mapping.

($\alpha = 1, 2$); we also define $r_\alpha := n_\alpha - k_\alpha$. The classical check (data) bits are labeled as $c_\alpha^{i_\alpha}$ ($v_\alpha^{j_\alpha}$), where $i_\alpha \in [r_\alpha]$ ($j_\alpha \in [n_\alpha]$), such that $H_\alpha^{i_\alpha j_\alpha} = 1$ if the $i_\alpha$th check $c_\alpha^{i_\alpha}$ has support on the $j_\alpha$th data bit $v_\alpha^{j_\alpha}$ and $H_\alpha^{i_\alpha j_\alpha} = 0$ otherwise. The product of these two classical codes yields the HGP code as illustrated in Fig. 2(a), where $H_1$ and $H_2$ represent the horizontal and vertical edges of the Tanner graph, respectively. The data qubits are labeled as $v_1^{j_1} v_2^{j_2}$ (lower-left quadrant with respect to the gray dashed lines) and $c_1^{i_1} c_2^{i_2}$ (upper-right quadrant), the $Z$-type syndrome qubits are labeled as $v_1^{j_1} c_2^{i_2}$ (upper-left quadrant), and the $X$-type syndrome qubits are labeled as $c_1^{i_1} v_2^{j_2}$ (lower-right quadrant). The HGP code has the parity check matrices

$$\begin{aligned} H_Z &= \left( H_2 \otimes I_{n_1} \mid I_{r_2} \otimes H_1^T \right), \\ H_X &= \left( I_{n_2} \otimes H_1 \mid H_2^T \otimes I_{r_1} \right), \end{aligned} \tag{1}$$

where $I_n$ denotes the $n$ by $n$ identity matrix, such that each $Z$-type check corresponding to $v_1^{j_1} c_2^{i_2}$ is supported by data qubit $v_1^{j_1} v_2^{j_2}$ ($c_1^{i_1} c_2^{i_2}$) if and only if $H_2^{i_2 j_2} = 1$ ($H_1^{i_1 j_1} = 1$), and similar holds for $X$-type checks. The code parameters are given by

$$[[n = n_1 n_2 + r_1 r_2,\ k = k_1 k_2,\ d = \min(d_1, d_2)]].$$

As a family of classical LDPC codes typically has constant encoding rate and distance rate, i.e., $k_\alpha / n_\alpha = \Theta(1)$ and $d_\alpha / n_\alpha = \Theta(1)$, the rates of the family of corresponding HGP codes scale as $k/n = \Theta(1)$ and $d/n = \Theta(1/\sqrt{n})$.

Following the convention from the literature [22, 27, 50, 52], we use the classical LDPC codes where each data bit is supported by 3 checks and each check supports 4 data bits. This yields a family of HGP codes with a fixed encoding rate $k/n = 1/25$, where each data (syndrome) qubit is connected to 6 to 8 syndrome (data) qubits. The HGP codes are known to perform well when the underlying classical codes have a large girth, i.e., the length of the shortest cycle in the Tanner graph [22, 38]. We implement the method in Ref. [21] to optimize the

distance and girth of classical LDPC codes in the `generate_ldpc` function (feature of `QUITS` outside the shaded area in Fig. 1).

### 3.2 Quasi-cyclic lifted product (QLP) codes

Lifted product codes are constructed by combining the hypergraph product with the lifting procedure [33, 34]. They encode a larger number of logical qubits with a larger code distance compared to HGP codes with a similar number of data qubits, as the lifting procedure reduces the symmetries of the product structure [8].

Here we consider the QLP codes [34, 38], which are obtained by taking the product of two base matrices defined over the quotient polynomial ring $\mathbb{R}[x]/(x^l-1)$ and then "lifting" each monomial entry into a $l$ by $l$ circulant matrix, where $l$ is the lift size [52]. Specifically, given the base matrices $B_\alpha$ of size $m_\alpha$ by $n_\alpha$ ($\alpha = 1, 2$), the parity check matrices of the QLP code are given by

$$\begin{aligned} H_Z &= \left(B_2 \otimes I_{n_1} \mid I_{m_2} \otimes B_1^T\right), \\ H_X &= \left(I_{n_2} \otimes B_1 \mid B_2^T \otimes I_{m_1}\right). \end{aligned} \tag{2}$$

Then, each monomial entry $x^s$ ($s \in [l]$, $x^l = 1$) is replaced by $l$ by $l$ matrix $M$ with entries $M^{ij} = \delta_{i,(j+s) \bmod l}$, where $\delta$ is the Kronecker delta symbol, and each zero entry is replaced by $l$ by $l$ matrix with all-zero entries [52]. Note that if $B_\alpha^{i_\alpha j_\alpha} = x^s$, then $(B_\alpha^T)^{j_\alpha i_\alpha} = x^{(l-s)}$. The number of data qubits is given by $n = l(n_1 n_2 + m_1 m_2)$ and the number of logical qubits is upper bounded as $k \leq n - l(n_1 m_2 + n_2 m_1)$.

The Tanner graph of a QLP code can also be understood from its product structure [52], as shown in Fig. 2(b). The data qubits are assigned labels $(v_1^{j_1} v_2^{j_2}, s)$ (lower-left quadrant) and $(c_1^{i_1} c_2^{i_2}, s)$ (upper-right quadrant), the Z-type syndrome qubits are assigned $(v_1^{j_1} c_2^{i_2}, s)$ (upper-left quadrant), and the X-type syndrome qubits are assigned $(c_1^{i_1} v_2^{j_2}, s)$ (lower-right quadrant), where $i_\alpha \in [m_\alpha]$, $j_\alpha \in [n_\alpha]$, and $s \in [l]$. Each $Z$-type check corresponding to $(v_1^{j_1} c_2^{i_2}, s_z)$ is supported by data qubit $(v_1^{j_1} v_2^{j_2}, s)$ [$(c_1^{i_1} c_2^{i_2}, s)$] if and only if $B_2^{i_2 j_2} = x^{(s_z - s) \bmod l}$ [$(B_1^T)^{i_1 j_1} = x^{(s - s_z) \bmod l}$], and similar holds for $X$-type checks. An example of unraveling the Tanner graph edge between two "lifted" nodes is visualized in Fig. 2(c).

The family of QLP codes used in this work is defined from the 3 by 5 base matrices in Eqs. (5) and (6) of Ref. [52], such that each data (syndrome) qubit is connected to 6 to 10 syndrome (data) qubits. Other base matrices can be straightforwardly incorporated.

### 3.3 Balanced product cyclic (BPC) codes

Balanced product codes are another example of codes constructed by reducing the symmetries of the hypergraph product [7]. Given two classical codes with a common symmetry group, the balanced product factors out the action of the group on their hypergraph product [8].

This work considers the family of BPC codes introduced in Ref. [47], although other BPC codes can be straightforwardly implemented. The codes are obtained from the balanced product $\otimes_H$ of two copies of a cyclic group $C_{3q} = \langle x \rangle$, where a smaller cyclic subgroup $H = C_q = \langle x^3 \rangle$ is factored out. The size of cyclic group $l := 3q$ can be considered as the lift size. Given two polynomials $p_1(x)$ and $p_2(x)$, the parity check matrices of the BPC code are given by

$$\begin{aligned} H_Z &= \left(e \otimes_H p_2(x) \mid (p_1(x) \otimes_H e)^T\right), \\ H_X &= \left(p_1(x) \otimes_H e \mid (e \otimes_H p_2(x))^T\right), \end{aligned} \tag{3}$$

where $e$ is the identity element. The balanced product $\otimes_H$ of two polynomials is a 3 by 3 matrix with polynomial entries. The specific rules of $\otimes_H$ can be found in Ref. [47]; crucially, $p_1(x) \otimes_H e$ only involves terms $x^s$ where $s \equiv 0 \bmod 3$, and $e \otimes_H p_2(x)$ is simply a diagonal matrix with $p_2(x)$ at all entries. Note that transpose works analogously to QLP codes. Then, each term $x^s$ ($s \in [l]$, $x^l = 1$) is lifted to $l$ by $l$ matrix $M$ with entries $M^{ij} = \delta_{i,(j+s) \bmod l}$, and each zero entry is lifted to $l$ by $l$ matrix with all-zero entries. This construction yields a family of codes with parameters [47]

$$[[n = 18q,\ k = 8,\ d \leq 2q]].$$

Although the encoding rate $k/n$ approaches zero asymptotically, the distance rate $d/n$ is relatively high at least for known examples [47]. Also, if each of $p_1(x)$ and $p_2(x)$ has 3 terms, each data (syndrome) qubit is connected to only 6 syndrome (data) qubits, making this family of codes a promising candidate for implementation.

The Tanner graph of the BPC codes is visualized in Fig. 2(d) and (e). In Fig. 2(d), the qubits are arranged such that each node (containing $l$

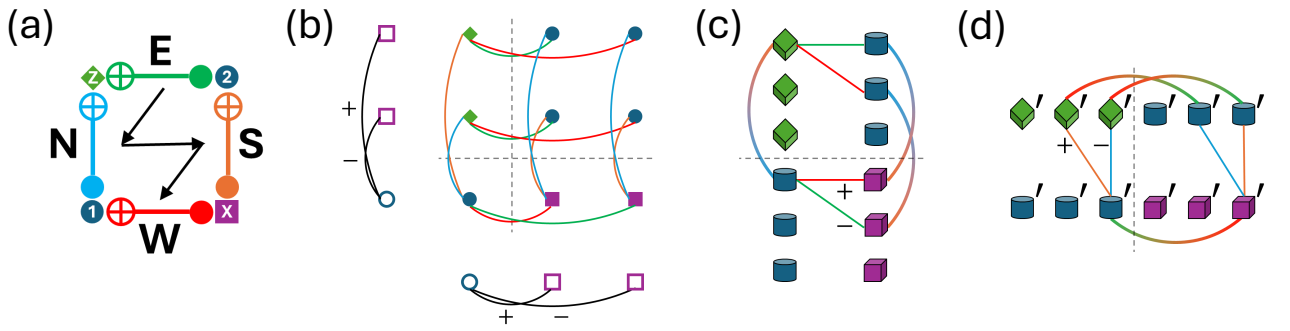



Figure 3: **Syndrome extraction circuit. (a)** A *qubit 4-cycle*, or a cycle of (data qubit)-(Z syndrome qubit)-(data qubit)-(X syndrome qubit), should consist of edges of all 4 directions. We use the convention in Ref. [50] and set the ordering of the directions as E→N→S→W. **(b)** Assignment of the signs of the classical Tanner graph's edges, which determines the directions of the quantum Tanner graph's edges. Note that the colors represent the directions as matched in (a) and are irrelevant to the edge coloration. **(c)(d)** Directions of the edges in a BPC code's Tanner graph. Horizontal edges are assigned directions E and W in (c), and vertical edges are assigned directions N and S in (d).

qubits) represents the row or column of the parity check matrices of Eq. (3), which are 3 by 6 matrices with polynomial entries. Specifically, the $j$th node of the lower-left [upper-right] quadrant is assigned the $j$th [$(j+3)$th] column that represents the data qubits ($j = 1, 2, 3$), and the $i$th node of the upper-left (lower-right) quadrant is assigned the $i$th row of $H_Z$ ($H_X$) that represents the $Z(X)$-type syndrome qubits ($i = 1, 2, 3$). As $e \otimes_H p_2(x)$ is diagonal, the *vertical* edges enjoy a one-to-one mapping between the nodes; i.e., each $Z(X)$-type check corresponding to a qubit in the $i$-th node of the upper-left (lower-right) quadrant is supported by data qubits in the $i$-th node of the lower-left (upper-right) quadrant.

The $3l$ qubits of the 3 nodes in each quadrant can be "shuffled" to yield the Tanner graph in Fig. 2(e), where the *horizontal* edges have a one-to-one mapping. Specifically, the $s$th qubit of the $i$th node ($s \in [l]$, $i \in [3]$) is shuffled to the $s'$th qubit of the $i'$th node, where $s = 3(m-1) + r$ ($m \in [q]$, $r \in [3]$), $s' = q(i-1) + m$, and $i' = r$. This allocation of qubits guarantees that the $Z(X)$-type check corresponding to a qubit in the $i'$-th node of the upper-left (lower-right) quadrant is supported by data qubits in the $i'$-th node of the upper-right (lower-left) quadrant, as $p_1(x) \otimes_H e$ matrix only contains terms of which power is multiple of 3. As will be shown in Sec. 4, the allocation of qubits that allows one-to-one mapping for edges in the horizontal or vertical direction is crucial to the design of the syndrome extraction circuit.

The family of BPC codes used in this work is defined from the weight 3 polynomials in Table 1 of Ref. [47], such that each data (syndrome) qubit is connected to 6 syndrome (data) qubits. Codes constructed from polynomials of larger weight or other cyclic groups can be straightforwardly implemented.

## 4 Syndrome extraction circuits

The syndrome extraction circuits of the QLDPC codes in Sec. 3 can be constructed in a unified way. We use a variant of the cardinal circuit developed in Ref. [50], which performed the first circuit-level simulations of HGP codes. Our method can be applied to QLP codes [52] and BPC codes, while also reducing the depth of the syndrome extraction circuit of small HGP codes (see Appx. A). The syndrome extraction depths of all QLDPC codes considered in this work are shown in Appx. A.

A syndrome extraction circuit, which is performed at each QEC cycle, consists of the following steps: (i) prepare the $Z(X)$-type syndrome qubits at the $|0\rangle$ ($|+\rangle$) state, (ii) apply the layers of CNOT gates sequentially, and (iii) measure the syndrome qubits in the respective basis. In each entangling gate layer, a qubit can have at most one gate applied to it. For optimal QEC performance, the application of CNOT gates needs to be parallelized as much as possible, reducing the number of layers.

The entangling gates are parallelized in the cardinal circuit of the HGP codes as follows [50]. First, a direction (E, N, S, W) is assigned to each

edge of the Tanner graph, where E or W (N or S) is assigned to a horizontal (vertical) edge. Each cycle of (data qubit)-($Z$ syndrome qubit)-(data qubit)-($X$ syndrome qubit), hereafter referred to as *qubit 4-cycle*, should consist of edges of all directions, as shown in Fig. 3(a). Then, for each direction, we apply *edge coloration* to the subgraph of the Tanner graph that consists of edges of the given direction, such that no two edges that share a vertex have the same color. The CNOT gates corresponding to edges with the same direction and color are applied simultaneously. Thus, the total number of entangling gate layers, or *syndrome extraction depth*, is given by the sum of the number of colors for each direction. The ordering of directions is crucial for extracting syndromes correctly. An improper ordering can entangle syndrome qubits and lead to random syndrome outcomes (see Appx. B of Ref. [16]). Here, we follow Ref. [50] and determine the ordering as E $\to$ N $\to$ S $\to$ W; i.e., all gates of direction E are applied, then N, S, and W.

Now we explain how the horizontal edges are assigned directions E or W, such that each qubit 4-cycle consists of both E and W edges. Vertical edges are analogously assigned directions N or S; see Fig. 3(b) for visualization. For HGP codes, data qubits $v_1^{j_1} v_2^{j_2}$, Z-type syndrome qubits $v_1^{j_1} c_2^{i_2}$, data qubits $c_1^{i_1} c_2^{i_2}$, and X-type syndrome qubits $c_1^{i_1} v_2^{j_2}$ are allocated in the lower-left, upper-left, upper-right, and lower-right quadrants, respectively. Thus, for each edge between nodes $v_1^{j_1}$ and $c_1^{i_1}$ of the underlying classical code's Tanner graph, the horizontal edges between qubits $v_1^{j_1} v_2^{j_2}$ and $c_1^{i_1} v_2^{j_2}$ in the lower two quadrants and the horizontal edges between qubits $v_1^{j_1} c_2^{i_2}$ and $c_1^{i_1} c_2^{i_2}$ in the upper two quadrants should have opposite directions, for all connected pairs of $v_2^{j_2}$ and $c_2^{i_2}$. To enforce this restriction, we assign a sign ($-$ or $+$) to each edge in the Tanner graph of the underlying classical code. Then, for each $-$ ($+$) edge, we assign direction E (W) to the corresponding horizontal edges of the quantum code's Tanner graph between the lower two quadrants. The edges between the upper two quadrants are automatically assigned the opposite direction.

To minimize the syndrome extraction depth, a balance between the directions is desired [50]; for each vertex, the horizontal (vertical) incident edges need to be distributed to E and W (N and S) directions as evenly as possible. We use a heuristic algorithm introduced in Appx. A to balance the signs of the classical Tanner graph's edges incident to each vertex. A key difference of our method from Ref. [50] is that the assignment of signs does not involve the positions of the vertices (after permutation). This flexibility allows for finding balanced solutions more easily, leading to a reduction in the syndrome extraction depth of $[[225, 9, 6]]$ HGP code from 12 to 8. We note that Ref. [50] defines the balanced ordering of the classical Tanner graph's vertices; however, an explicit method for finding such an ordering is not provided.

For QLP and BPC codes, which also have a product structure, the qubits are similarly allocated to four quadrants, as illustrated in Fig. 2. Thus, the same procedure as described above assigns the direction to each edge between two lifted nodes, each of which represents a set of $l$ qubits. The edges between the two sets of $l$ qubits in the quantum code's Tanner graph [see Fig. 2(c)] are assigned the same direction as the corresponding edge between the lifted nodes.

The balanced product structure in BPC codes requires additional care in assigning the directions, as shown in Fig. 3(c) and (d). In (c), the horizontal edges below the dashed line can be thought of as the edges of a "classical" Tanner graph, to which signs are assigned. Due to the one-to-one mapping of the vertical edges, the pair of horizontal edges is unambiguously defined for each qubit 4-cycle. Thus, E and W directions can be assigned using the same method. In (d), the vertical edges between the "shuffled" nodes are assigned directions analogously, which is again enabled by the one-to-one mapping of the horizontal edges. This completes the assignment of directions such that each qubit 4-cycle has edges of all 4 directions.

## 5 Decoder implementation

In this section, we introduce a general sliding-window decoding framework [27]. For decoding under a circuit-level noise model, we enable implementing various inner decoders within the sliding-window decoding framework, as illustrated in Fig. 1. The modular structure of `QUITS` that decouples the inner decoder from the sliding-window decoding framework facilitates the development and testing of various custom-defined in-

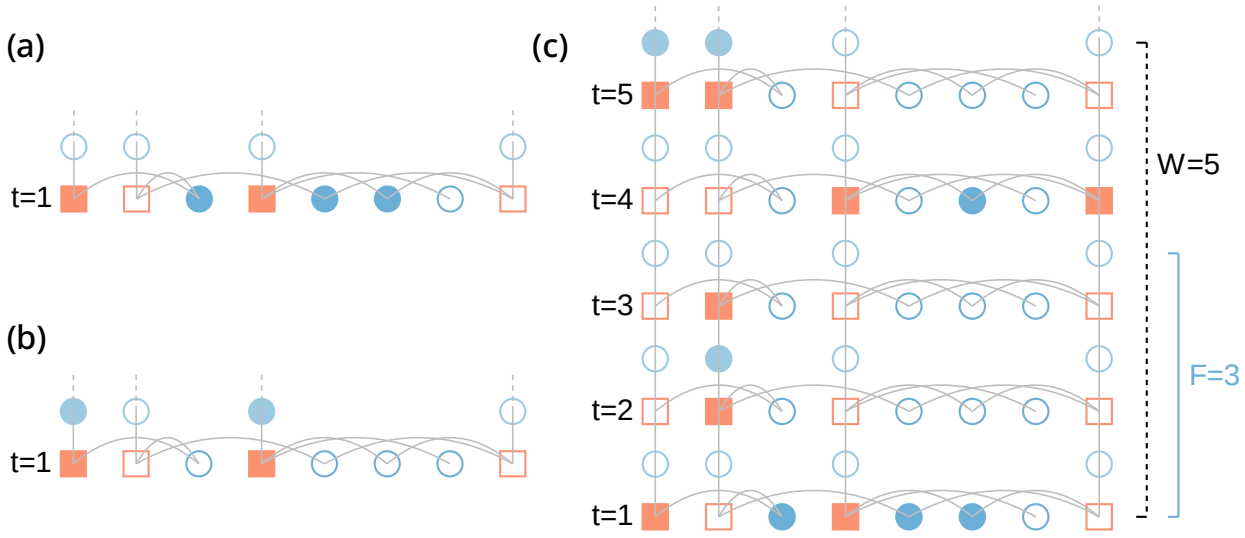



Figure 4: **Space-time decoding graph.** Tanner graph constructed from a detector error matrix. Each square represents a detector. Each circle represents a possible error mechanism. Filled circles are errors that occur. Filled squares are the detectors flipped by the errors. For simplicity, we show a phenomenological noise model where there are only data qubit errors (blue circles) and measurement errors (light blue circles on the vertical edges). Dashed gray lines mark the time-like boundary of the decoding graph, which connects error mechanisms to detectors of future time steps they support. **(a)(b)** Two possible combinations of errors that create the same detector flips. If single-shot error correction is performed and the error shown in (a) occurs, a minimum-weight decoder will apply the correction shown in (b) as it is the error combination with the lowest weight that produces the given flipped detector set. **(c)** A decoding window for a $(5, 3)$ sliding-window decoding. For each decoding window, we pass the detector flips of $W$ consecutive time steps to an inner decoder. Based on the correction suggested by the inner decoder, we apply the correction that flips the detectors in the first $F$ time steps. By increasing the size of the decoding window, more detector information is included in the decoding window. Thus, the decoder can correct the error shown in (a) that occurs at the beginning of the decoding window.

ner decoders.

### 5.1 Detectors

Before we discuss sliding-window decoding, we first introduce detectors. Detectors are products of measurements that return $+1$ in the absence of errors, and $-1$ when errors are detected. As each detector can only be flipped by certain combinations of physical errors, a decoder can locate and correct errors based on the detector values. We construct $P$-type *detectors* $D_P^i(t)$ at round $t$ by taking the parity of two consecutive measurements $M_P^i(t)$ at round $t$ and $M_P^i(t+1)$ at round $t+1$ of the $i$th $P$-type check, i.e.,

$$D_P^i(t) := M_P^i(t) M_P^i(t+1), \tag{4}$$

where $i$ denotes the index of the detector. For the initial round $t = 0$, we define the detectors to be $D_P^i(t=0) := M_P^i(t=1)$.

Similarly to parity check matrices, we construct *detector error matrices*. In a $P$-type detector error matrix $\widetilde{H}_P$ for the whole circuit, each column represents a possible error mechanism that flips $P$-type detector, while each row represents a $P$-type detector. Each error mechanism is an incident of error, such as qubit error, measurement error, and gate error, that can occur independently. Two error mechanisms are equivalent if they generate the same syndromes and logical observables. A matrix element $\widetilde{H}_P^{ij} = 1$ if the $i$th detector is flipped by the $j$th error mechanism. The detector error matrices can be readily converted from the detector error model in Stim [17, 18, 25]. When constructing the detector error matrix, equivalent error mechanisms are merged into a single error mechanism with an updated error rate. An example Tanner graph constructed from a detector error matrix is shown in Fig. 4(c).

There are two categories of error mechanisms

captured by detector error matrices: (i) space-like errors, which only flip the detectors of the same round (e.g. the blue circles in Fig. 4(c)), and (ii) time-like errors, which flip the detectors in two consecutive rounds (e.g. the light blue circles on the vertical edges in Fig. 4(c)). In a phenomenological noise model, which includes only data qubit errors and measurement errors, space-like errors are the data qubit errors, and time-like errors are the measurement errors. In a circuit-level noise model, where errors may occur during the execution of the syndrome extraction circuit, idling errors on data qubits can also cause time-like errors. For example, if an idling $X$-type error on a data qubit occurs between the CNOT gates in the syndrome extraction circuit at round $t$, only a subset of the $Z$-type checks supported by this data qubit will have their signs flipped at round $t$, and the remaining checks' signs will be flipped at round $t+1$, together causing a time-like error. Similarly, gate errors can cause both space-like and time-like errors, whereas measurement errors can only cause time-like errors. To achieve accurate decoding, using a decoding graph that includes both space-like and time-like errors is necessary, as described in Fig. 4. Note that a single error mechanism cannot flip detectors from more than two rounds.

## 5.2 Sliding-window decoding

Due to the existence of time-like errors, multiple rounds of syndrome measurement results need to be decoded together, which necessitates sliding-window decoding. While some QLDPC codes support single-shot error correction, where error correction is performed immediately after each round of syndrome extraction [11, 13, 14], sliding-window decoding has been shown to outperform single-shot error correction for HGP and QLP codes even under a phenomenological noise model [27]. The performance degradation of single-shot error correction arises from the decoder's inability to distinguish between space-like errors and time-like errors. For example, Fig. 4(a) and (b) present two example error combinations that generate the same syndrome in a phenomenological noise model. Given only one round of syndrome extraction, if independent measurement errors and data qubit errors occur at the same probability, a minimum-weight decoder always chooses to apply the error combination in (b) as it contains fewer independent errors.

Sliding-window decoding is capable of distinguishing space-like and time-like errors. In general, sliding-window decoding is implemented by dividing syndromes from different rounds into overlapping decoding windows that are individually passed to an inner decoder. Only the correction for the portion of the window that does not overlap with the next is applied, while the syndromes in the next decoding window are updated accordingly [27]. An example of a decoding window under a phenomenological noise model with $W = 5$ rounds of syndrome extraction is shown in Fig. 4(c). In this example, errors at the beginning of the window can be reliably corrected. Note that data-qubit errors occurring at the end of the window cannot be distinguished from measurement errors, for the same reason as the case in Fig. 4(a) and (b). To ensure reliability, only the correction for the first $F$ rounds of syndrome is applied, while the remaining syndrome corrections are deferred to the next decoding window. Fig. 4(c) shows an example of $F = 3$.

Formally, we parametrize space-time sliding-window decoding by $(W, F)$, where $W$ denotes the number of detector rounds included in each window and $F$ denotes the number of rounds in correction that we commit [27]. To decode the $w$th window, we pass into the inner decoder a submatrix of the $P$-type detector error matrix consisting of rows labeled from $(w-1)Fc$ to $((w-1)F+W-1)c$ and columns that have non-trivial support on these rows, where $c$ denotes the number of $P$-type checks we measure per round and $w = 1, \cdots, N$ with $N$ as the total number of decoding windows. The inner decoder produces a correction based on these submatrices and the corresponding error rates. Before the last window, from the correction, the error mechanisms affecting $D_P^i(t)$ where $i = 1, \cdots, c$ and $t = (w-1)F, \cdots, wF-1$ are committed. The following set of detectors $\{D_P^i(wF)\}$ is updated accordingly. In the last window, all the correction is applied. See also Refs. [2, 18, 30, 44] where a similar decoding scheme is applied to other QLDPC codes.

Before discussing inner decoders, we highlight that `QUITS` features a modular structure that separates inner decoders from the space-time sliding-window decoding framework, unlike previous im-

plementations where the inner decoder is tightly integrated into the framework [18]. This separation enables convenient testing of different inner decoder algorithms.

### 5.3 Inner Decoder

While the outer sliding-window decoder characterizes the relationship between possible error mechanisms and the detectors, we must still decode the detector error matrix within a window. One common class of decoders that can be applied to infer an error pattern is the belief-propagation(BP)-based decoders [26, 33].

#### 5.3.1 Belief Propagation Decoding

The BP decoder is an iterative message-passing algorithm widely used in classical error correction, particularly for LDPC codes. When applied to the Tanner graph representing the detector error matrix, BP iteratively exchanges messages between the detectors and the error mechanisms, attempting to identify a probable error pattern that matches the observed detector flips. If BP successfully finds such an error pattern, it is said to *converge*. The computational complexity of BP decoding is $\mathcal{O}(\mathcal{I}n)$, where $\mathcal{I}$ denotes the number of message passing iterations.

When applied to quantum codes, BP decoding faces several known challenges that degrade its performance. In particular, short cycles in the factor graph can hinder BP convergence. Quantum LDPC codes inherently contain many loops due to degeneracy, making the factor graph far from locally tree-like and obstructing the convergence of BP. Under circuit-level noise, this issue is further amplified by correlated errors introduced around quantum gates. A single physical error can propagate through two-qubit gates and trigger multiple syndrome measurements, effectively creating additional short cycles in the Tanner graph of the detector error model. Another challenge is the presence of syndrome measurement errors, where a single measurement error can activate two consecutive detectors [18].

#### 5.3.2 Heuristics for improving BP

Motivated by issues of BP on quantum Tanner graphs, some post-processors of BP have been proposed. In practice, these BP-based post-processing decoders substantially improve reliability in the presence of correlated fault patterns, which is crucial for decoding QLDPC codes under realistic circuit-level noise. In this section, we briefly review some of these post-processing schemes.

In cases where BP does not converge, it still produces soft estimates on the likelihoods of all the error mechanisms. In this case, order-statistics decoding (OSD) can be applied as a post-processing step to improve the BP decoding performance. This approach, known as BP with order-statistics decoding (BP-OSD), takes the BP soft output and computes a probable error pattern that matches the detector flips by solving a system of linear equations. Even a low-order OSD post-processing yields significant gains. BP-OSD with order zero has a computational complexity of $\mathcal{O}(n_e^3)$, where $n_e$ denotes the number of error mechanisms. For a more comprehensive introduction to BP-OSD, we refer the reader to Ref. [33].

Another decoder supported by `QUITS`, imported from [41], is belief propagation with localized statistics decoding (BP-LSD) [26], a more recent decoding scheme that improves upon BP-OSD. To improve the efficiency of OSD, BP-LSD identifies small, isolated clusters consisting of check nodes and fault nodes in the Tanner graph and then decodes them separately using OSD in parallel. Clusters are grown starting from the activated detector nodes, with their growth guided by BP. They are merged when necessary and expanded until a local solution can be identified. BP-LSD has been shown to achieve the same level of accuracy as BP-OSD while enabling parallel decoding of the cluster solutions.

In `QUITS`, we provide built-in support of the BP-OSD and BP-LSD decoders from the LDPC package [41] within the space-time sliding-window framework, alongside a general decoding framework that accommodates any customized decoders.

## 6 Results

In this section, we simulate the circuit-level performance of the memory of various QLDPC codes. The HGP codes used in this section are constructed from the product of classical codes, each associated with a random $(3, 4)$-regular Tanner graph. These classical codes are obtained

from the procedure described in Ref. [22], implemented in the `generate_ldpc` function of `QUITS`. By sampling the underlying classical codes with a minimum girth 6, we choose HGP codes with the largest code distance. We denote these codes by $n$ as HGP225, HGP625, and HGP900. With `QUITS` circuit generator, we found circuits with minimum syndrome extraction depth $\min d_c = 8$ for HGP225 and 12 for HGP625 and HGP900. The QLP and BPC codes are brought from Refs. [52] and [47], respectively, as explained in Sec. 3. The minimum syndrome extraction depth found using `QUITS` was $d_c = 12$ and 8 for the QLP and BPC code family, respectively. See Appx. A for a detailed discussion of the syndrome extraction depth $d_c$.

For the noise model, we use the standard circuit-level depolarizing noise model characterized by the physical error rate $p$. At each time step, all qubits experience a single-qubit depolarizing channel where all single-qubit Pauli errors $X$, $Y$, and $Z$ occur at an equal probability $p/3$ when idling or subjected to single-qubit gates. All the CNOT gates induce two-qubit depolarizing channels on the involved qubits, where the 15 possible two-qubit Pauli errors from the set $\{I, X, Y, Z\}^{\otimes 2}/I \otimes I$ occur at an equal probability $p/15$. Reset and measurement errors on syndrome qubits for $Z(X)$-type error detection are modeled by single qubit bit-flip (phase-flip) channels where $X(Z)$ errors occur at a probability $p$. Each of these single-qubit or two-qubit Pauli errors at different locations represents an error mechanism. Independent errors on syndrome qubits may propagate through CNOT gates, leading to correlated errors on multiple data qubits. These correlated errors, known as hook errors, can significantly degrade the code performance. [13, 49].

For each code, we perform $T = 16$ rounds of noisy syndrome extraction circuits. At the end of the circuit, each physical qubit undergoes a noiseless measurement, with the results combined into the final stabilizer and logical operator measurements. We vary the number of samples with the physical error rate $p$ to obtain similar error bars for each $p$. We calculate the logical failure rate as $\mathrm{LFR}(p) = 1-(1-p_L)^{1/T}$ [52], where $p_L$ is the rate at which a logical error occurs at any logical qubit at any round. We report the LFRs for $X$-type logical errors only; $Z$-type logical errors can be simulated similarly.

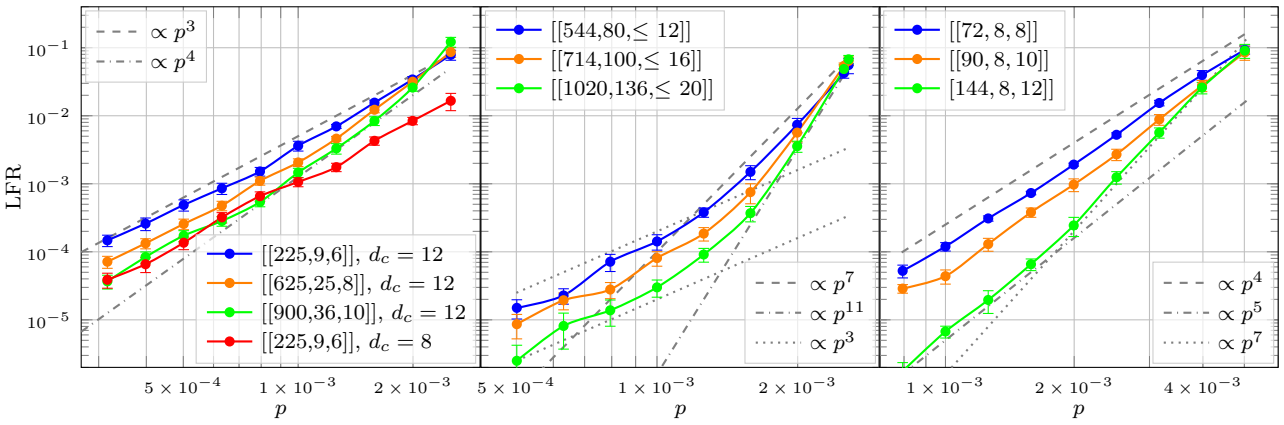
## 6.1 Threshold plots

We present our results in Fig. 5. All simulations are performed using ($W = 5$, $F = 3$) sliding-window decoding with the BP-OSD decoder [42] as the inner decoder. We set the BP-OSD parameters as $(\mathcal{I}, O) = (10, 1)$ for the HGP and QLP codes and $(20, 10)$ for the BPC codes, where $\mathcal{I}$ is the maximum number of BP iterations and $O$ is the OSD order. We expect the results can be further improved by optimizing the decoding parameters; see Sec. 6.2 for a detailed discussion.

In Fig. 5(a), we plot the LFRs of all three HGP codes implemented with syndrome extraction circuits of $d_c = 12$. A threshold is identified at approximately 0.23%. Comparing our results for HGP625 with those in Ref. [50], which uses the single-shot decoder based on BP and small-set flip [22], the LFR values of our simulations are approximately an order of magnitude lower, whereas the threshold value is on par.

In addition to the threshold simulation, the red curve in Fig. 5(a) shows the results for the HGP225 code with a $d_c = 8$ circuit, found by the method described in Sec. 4. This circuit saturates the lower bound of $d_c$ derived in Ref. [50] for this family of codes; see Appx. A. HGP225 with $d_c = 8$ outperforms all three HGP codes with $d_c = 12$ near the threshold, even though HGP625 and HGP900 have higher code distances. Even for $p \approx 3 \times 10^{-4}$, the performance of HGP225 with $d_c = 8$ is on par with that of HGP900 with $d_c = 12$. This significant improvement in the logical performance arises from the fact that fewer error locations exist in the circuit with lower $d_c$.

The slope of the error-rate curve represents $\lfloor (d_{\text{eff}} + 1)/2 \rfloor$, where $d_{\text{eff}}$ is the *effective distance* of the code, which indicates that the code can correct up to approximately $(d_{\text{eff}} - 1)/2$ independent errors. Note that here we do not consider methods that use post-selection to discard shots for which two minimum-weight predictions of physical errors have the same weight and correspond to different logical classes, which is common in even-distance codes [32, 39]. Before analyzing the results, we briefly explain how the slope may depend on the error rate $p$. For an ideal decoder that is guaranteed to correct all error configurations of weight $\lfloor (d-1)/2 \rfloor$, such as the minimum-weight perfect matching decoder of the




(a) HGP codes simulation (b) QLP codes simulation (c) BPC codes simulation

Figure 5: **LFR simulation.** Logical memory simulations of various QLDPC codes under the standard circuit-level depolarizing noise model using a $(5, 3)$ space-time sliding-window decoding with a BP-OSD inner decoder. The error bars represent the 95% confidence interval. Reference curves are drawn for the comparison of effective distances $d_{\text{eff}}$. **(a)** HGP code simulation. A threshold is identified at approximately $0.23\%$ for circuits with syndrome extraction depth $d_c = 12$. Note that the $[[225, 9, 6]]$ HGP code with $d_c = 8$ outperforms all HGP codes implemented with a $d_c = 12$ circuit in the high-error region and exhibits comparable performance to the $[[900, 36, 10]]$ code with a $d_c = 12$ circuit as $p$ decreases. **(b)** QLP code simulation. A threshold is identified at approximately $0.24\%$. Notably, an error floor behavior is observed for the QLP code. This is evident as the slopes of the curves reduce, showing no clear dependence on the code size. This error floor is likely caused by small cycles in the space-time detector error matrix, which hinder the convergence of BP. **(c)** BPC code simulation. A threshold is observed at approximately $0.47\%$. The slopes of the curves suggest that our circuits maintain $d_{\text{eff}}$ comparable to $d$ for the simulated range of $p$.

surface code, the slope is $\lfloor (d+1)/2 \rfloor$ for $\forall p$ that is sufficiently below the threshold. However, more heuristic decoders such as BP may fail to decode a few specific error configurations of relatively low weight, especially in the circuit level. The number of error configurations of weight $w$ that lead to a logical failure is denoted as $A(w)$, where $A(w)$ typically increases with $w$. Then, the probability of logical failure due to an error of weight $w$ is given by $A(w)p^w$. At relatively high $p$, the total LFR may be dominated by $A(w)p^w$ of larger $w$, due to the larger value of $A(w)$. However, as $p$ is decreased, $A(w)p^w$ of a smaller $w$ value eventually dominates, as $p^w$ is larger for smaller $w$. Thus, as $p$ is decreased, the slope $\lfloor (d_{\text{eff}} + 1)/2 \rfloor$ saturates to a lower value, which represents the minimum weight of errors that cause a logical failure. A reduction from $d$ to $d_{\text{eff}}$ can result from hook errors in syndrome extraction circuits or the decoding process; see Sec. 7 for a detailed discussion and potential strategies to improve $d_{\text{eff}}$.

For an easy comparison of $d_{\text{eff}}$, in Fig. 5(a), we show two reference curves in gray. The slope of the HGP225 curve is near 3, which indicates that our circuit simulation achieves $d_{\text{eff}} \approx d$ for HGP225 in the simulated region of $p$. For HGP900, $d_{\text{eff}} \approx 7 < d$ near the threshold and is further reduced for lower values of $p$.

Figure 5(b) shows the results for the QLP codes. The codes we simulate here are introduced in Ref. [52]. The threshold is identified at approximately 0.24%. In the low-$p$ region, all QLP codes display comparable slopes, suggesting the reduction of $d_{\text{eff}}$ similarly to the HGP900 code. The slopes indicate that the lowest weight of uncorrectable errors in these codes is approximately the same, despite the different code distances $d$. Such error floor behavior is likely due to the difficulty of the BP-based decoder in handling small cycles in the detector error matrix. Similar behavior is observed for the simulation results in Ref. [52].

Figure 5(c) shows the results for the BPC codes introduced in Ref. [47]. This family of codes exhibits a relatively high circuit-level threshold at approximately 0.47% due to the low degree of connectivity. The slopes of the curves indicate that $d_{\text{eff}} \approx d$ for the simulated range of $p$, although error floors may still occur in the region of lower $p$ for BP-based decoders.

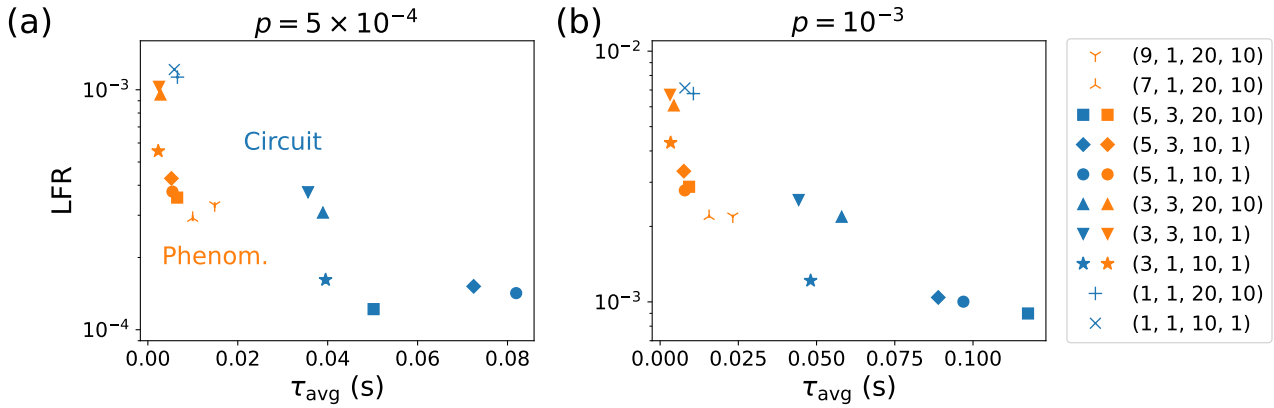



Figure 6: **Trade-offs between the decoding runtime and the LFR.** Logical memory of the HGP225 code with $d_c = 8$ circuit is simulated using various decoder parameters under the standard circuit-level depolarizing noise model where **(a)** $p = 5 \times 10^{-4}$ and **(b)** $p = 10^{-3}$. For all points, BP-OSD decoder is used as the inner decoder. We report the LFR for $T = 16$ rounds of noisy syndrome extraction circuit and the total decoding runtime divided by the number of times the inner decoder is called. The numbers in the legend represent $(W, F, \mathcal{I}, O)$ where $W$ is the width of the sliding window, $F$ is the offset between consecutive sliding windows, $\mathcal{I}$ is the maximum number of BP iterations, and $O$ is the OSD order. For blue symbols, detector error matrices for circuit-level noise are used. For orange symbols, detector error matrices for the phenomenological noise model are used. The runtimes, averaged over 20000 samples for each parameter set, are evaluated in a single node of the Duke Computing Cluster. Error bars are smaller than the symbols and thus not shown.

## 6.2 Trade-offs between time and accuracy

Even after the QEC code, the syndrome extraction circuit, and the inner decoder are determined, many degrees of freedom remain for implementing the decoder. In this section, we discuss the trade-offs between the decoding runtime and the LFR that are relevant for choosing the appropriate set of decoder parameters.

Figure 6 shows the results for simulating the logical memory circuit of the HGP225 code with syndrome extraction depth $d_c = 8$. The average decoding runtime $\tau_{\text{avg}}$ is given by $\tau/N$, where $\tau$ is the total runtime for decoding $T$ rounds and $N$ is the number of times the inner decoder is called. Note that we use the python version of the BP-OSD decoder [42] provided by the LDPC package [41] as the inner decoder; we expect that $\tau_{\text{avg}}$ can be overall reduced by using the C++ version [41]. Our goal here is to compare the trend of LFR with respect to $\tau_{\text{avg}}$ over various sets of parameters for the sliding window and the BP-OSD decoder.

We observe that there is an overall trend of decreasing LFR as $\tau_{\text{avg}}$ increases. This highlights that when evaluating the performance of a decoder, the accuracy and the runtime need to be reported together. For example, while the LFR can be further reduced by increasing the window size $W$ with fixed $F$, the detector error matrix would become too large for realistic implementation, especially for larger codes. Also, the optimal set of decoder parameters within a runtime constraint may depend on the physical error rate $p$. We anticipate that `QUITS` leads to a benchmark for a fair comparison among various decoders.

Among the decoder parameters considered here, increasing $W$ most significantly reduces the LFR at the cost of larger $\tau_{\text{avg}}$. The decoding runtime increases with $W$ due to the larger size of the detector error matrix slice for each decoding window. The LFR decreases with $W$, as a larger decoding window provides more syndrome information to better distinguish space-like and time-like errors that occur at the beginning of the window.

Improving both $\mathcal{I}$ and $O$ is necessary for achieving the lowest LFR; however, the overall performance improvement is minor compared to the effects of increasing $W$. Comparing the blue and orange symbols, using detector error matrices for circuit-level noise is essential for achieving low LFR, as explained in Sec. 5. Interestingly, with suitable choices of $(W, F, \mathcal{I}, O)$, using detector error matrices for the phenomenological noise model can achieve a similar LFR with a lower $\tau_{avg}$ compared to using detector error matrices for circuit-level noise. Future work may explore

the trade-offs for other decoder options, such as using parallelization for BP.

## 7 Conclusion and Outlook

In this work, we introduced `QUITS`, an open-source, modular framework for circuit-level simulation of quantum low-density parity-check (QLDPC) codes. `QUITS` enables end-to-end simulations by allowing users to flexibly combine various QEC codes, syndrome extraction circuits, decoding algorithms, and noise models. As illustrated in Fig. 1, the modular design of `QUITS` facilitates performance comparisons between different implementations by allowing users to easily switch or replace individual components, including the QLDPC code, the syndrome extraction circuit, the decoder, and the decoding window configuration.

Our framework supports several prominent QLDPC families, including hypergraph product (HGP) codes, quasi-cyclic lifted product (QLP) codes, and balanced product cyclic (BPC) codes. Thanks to its modular structure, `QUITS` can be readily extended to incorporate new code families and decoding algorithms. As part of the framework, we introduce a syndrome extraction circuit that applies across all three code families. Notably, for the HGP225 code, our circuit achieves a lower syndrome extraction depth than the original cardinal circuit in Ref. [50], resulting in smaller detector error matrices and improved logical failure rates.

By applying `QUITS` to simulate the circuit-level performance of various QLDPC codes, we observe a notable gap between the effective distance $d_{\text{eff}}$ and the original code distance $d$ in the low-error-rate regime, as shown in Sec. 6. This observation points to several directions for future investigation.

First, one potential cause of the reduced effective distance is the suboptimal performance of belief propagation (BP) decoding within the sliding-window framework. Specifically, when BP is applied to the Tanner graph of the detector error matrix, its performance can be degraded by certain subgraph structures known as trapping sets [40], which are known to cause decoding failures and give rise to error floors. One future direction is to investigate these failure mechanisms using `QUITS` by identifying and analyzing the trapping sets responsible for decoding breakdowns in circuit-level simulations. This may further guide our strategies to mitigate error floors and improve the effective distance, either by improving the BP decoding algorithm itself [28] or by finding a better Tanner graph of the detector error matrix.

Second, hook errors in the syndrome extraction circuit may also cause a reduction of the effective distance. If the high-weight hook errors overlap with the support of a logical observable, a logical error may occur from error mechanisms of weight lower than $\lfloor (d-1)/2 \rfloor$, resulting in the reduction of $d_{\text{eff}}$. For HGP codes, Refs. [31, 36] prove that hook errors do not reduce $d_{\text{eff}}$ in the cardinal circuit [50]; however, this proof does not apply to other QLDPC codes. A design principle for syndrome extraction circuits of QLP and BPC codes that suppresses hook errors and maximizes $d_{\text{eff}}$ is highly desired. The performance of such syndrome extraction circuits can be efficiently tested using our framework.

Finally, `QUITS` is currently limited to simulating the logical memory of QLDPC codes. A future direction is to enable simulations of logical operations, such as the joint measurement of logical observables [12, 52]. The circuit that performs the logical operation may be co-designed with the decoder implementation, such that $d_{\text{eff}}$ of the logical operation is not reduced compared to that of the logical memory, as recently demonstrated for the bivariate bicycle codes [12]. We hope that `QUITS` extends to providing an end-to-end framework for simulating logical operations of various QLDPC codes.

## 8 Acknowledgement

The authors thank Shilin Huang for the valuable discussions. This work was supported by ARO/LPS QCISS program (W911NF-21-1-0005), the NSF QLCI for Robust Quantum Simulation (OMA-2120757), and NSF Grant 2106213. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, ARO or LPS.

## References


[1] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pages 176–188, 1997. DOI: 10.1145/258533.258579.

[2] Lucas Berent, Timo Hillmann, Jens Eisert, Robert Wille, and Joschka Roffe. Analog information decoding of bosonic quantum low-density parity-check codes. PRX Quantum, 5(2):020349, 2024. DOI: 10.1103/PRXQuantum.5.020349.

[3] Michael E Beverland, Aleksander Kubica, and Krysta M Svore. Cost of universality: A comparative study of the overhead of state distillation and code switching with color codes. PRX Quantum, 2(2):020341, 2021. DOI: 10.1103/PRXQuantum.2.020341.

[4] Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. Logical quantum processor based on reconfigurable atom arrays. Nature, 626(7997):58–65, 2024. DOI: 10.1038/s41586-023-06927-3.

[5] Sergey Bravyi, David Poulin, and Barbara Terhal. Tradeoffs for reliable quantum information storage in 2d systems. Physical Review Letters, 104(5):050503, 2010. DOI: 10.1103/PhysRevLett.104.050503.

[6] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. Nature, 627(8005):778–782, 2024. DOI: 10.1038/s41586-024-07107-7.

[7] Nikolas P Breuckmann and Jens N Eberhardt. Balanced product quantum codes. IEEE Transactions on Information Theory, 67(10):6653–6674, 2021. DOI: 10.1109/TIT.2021.3097347.

[8] Nikolas P Breuckmann and Jens Niklas Eberhardt. Quantum low-density parity-check codes. PRX Quantum, 2(4):040101, 2021. DOI: 10.1103/PRXQuantum.2.040101.

[9] Madelyn Cain, Chen Zhao, Hengyun Zhou, Nadine Meister, J Pablo Bonilla Ataides, Arthur Jaffe, Dolev Bluvstein, and Mikhail D Lukin. Correlated decoding of logical algorithms with transversal gates. Physical Review Letters, 133 (24):240602, 2024. DOI: 10.1103/PhysRevLett.133.240602.

[10] A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. Phys. Rev. A, 54:1098–1105, Aug 1996. DOI: 10.1103/PhysRevA.54.1098.

[11] Earl T Campbell. A theory of single-shot error correction for adversarial noise. Quantum Science and Technology, 4 (2):025006, 2019. DOI: 10.1088/2058-9565/aafc8f.

[12] Andrew Cross, Zhiyang He, Patrick Rall, and Theodore Yoder. Improved qldpc surgery: Logical measurements and bridging codes. arXiv preprint arXiv:2407.18393, 2024. DOI: 10.48550/arXiv.2407.18393.

[13] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. Journal of Mathematical Physics, 43(9):4452–4505, 2002. DOI: 10.1063/1.1499754.

[14] Omar Fawzi, Antoine Grospellier, and Anthony Leverrier. Constant overhead quantum fault tolerance with quantum expander codes. Communications of the ACM, 64(1): 106–114, 2020. DOI: 10.1145/3434163.

[15] Austin G Fowler, Ashley M Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. Physical Review A, 80(5):052312, 2009. DOI: 10.1103/PhysRevA.80.052312.

[16] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. Physical Review A, 86(3):032324, 2012. DOI: 10.1103/PhysRevA.86.032324.

[17] Craig Gidney. Stim: a fast stabilizer circuit simulator. Quantum, 5:497, 2021. DOI: 10.22331/q-2021-07-06-497.

[18] Anqi Gong, Sebastian Cammerer, and Joseph M Renes. Toward low-latency iterative decoding of QLDPC codes under circuit-level noise. arXiv preprint arXiv:2403.18901, 2024. DOI: 10.48550/arXiv.2403.18901.

[19] Google Quantum AI and Collaborators. Quantum error correction below the surface

code threshold. Nature, 638(8052):920–926, 2025. DOI: 10.1038/s41586-024-08449-y.

[20] Daniel Gottesman. Theory of fault-tolerant quantum computation. Physical Review A, 57(1):127, 1998. DOI: 10.1103/PhysRevA.57.127.

[21] Antoine Grospellier. Constant time decoding of quantum expander codes and application to fault-tolerant quantum computation. PhD thesis, Sorbonne Université, 2019.

[22] Antoine Grospellier, Lucien Grouès, Anirudh Krishna, and Anthony Leverrier. Combining hard and soft decoders for hypergraph product codes. Quantum, 5: 432, 2021. DOI: 10.22331/q-2021-04-15-432.

[23] Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching. ACM Transactions on Quantum Computing, 3(3): 1–16, 2022. DOI: 10.1145/3505637.

[24] Oscar Higgott and Craig Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. Quantum, 9:1600, 2025. DOI: 10.22331/q-2025-01-20-1600.

[25] Oscar Higgott, Thomas C Bohdanowicz, Aleksander Kubica, Steven T Flammia, and Earl T Campbell. Improved decoding of circuit noise and fragile boundaries of tailored surface codes. Physical Review X, 13(3):031007, 2023. DOI: 10.1103/PhysRevX.13.031007.

[26] Timo Hillmann, Lucas Berent, Armanda O Quintavalle, Jens Eisert, Robert Wille, and Joschka Roffe. Localized statistics decoding for quantum low-density parity-check codes. Nature Communications, 16(1):8214, 2025. DOI: 10.1038/s41467-025-63214-7.

[27] Shilin Huang and Shruti Puri. Increasing memory lifetime of quantum low-density parity check codes with sliding-window noisy syndrome decoding. Physical Review A, 110(1):012453, 2024. DOI: 10.1103/PhysRevA.110.012453.

[28] Kenta Kasai. Efficient mitigation of error floors in quantum error correction using non-binary low-density parity-check codes. arXiv preprint arXiv:2501.13923, 2025. DOI: 10.48550/arXiv.2501.13923.

[29] Anthony Leverrier and Gilles Zémor. Quantum Tanner codes. In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), pages 872–883. IEEE, 2022. DOI: 10.1109/FOCS54457.2022.00117.

[30] Hsiang-Ku Lin, Xingrui Liu, Pak Kau Lim, and Leonid P Pryadko. Single-shot and two-shot decoding with generalized bicycle codes. arXiv preprint arXiv:2502.19406, 2025. DOI: 10.48550/arXiv.2502.19406.

[31] Argyris Giannisis Manes and Jahan Claes. Distance-preserving stabilizer measurements in hypergraph product codes. Quantum, 9: 1618, 2025. DOI: 10.22331/q-2025-01-30-1618.

[32] A Paetznick, MP da Silva, C Ryan-Anderson, JM Bello-Rivas, JP Campora III, A Chernoguzov, JM Dreiling, C Foltz, F Frachon, JP Gaebler, et al. Demonstration of logical qubits and repeated error correction with better-than-physical error rates. arXiv preprint arXiv:2404.02280, 2024. DOI: 10.48550/arXiv.2404.02280.

[33] Pavel Panteleev and Gleb Kalachev. Degenerate quantum LDPC codes with good finite length performance. Quantum, 5:585, 2021. DOI: 10.22331/q-2021-11-22-585.

[34] Pavel Panteleev and Gleb Kalachev. Quantum LDPC codes with almost linear minimum distance. IEEE Transactions on Information Theory, 68(1):213–229, 2021. DOI: 10.1109/TIT.2021.3119384.

[35] Balint Pato, Theerapat Tansuwannont, and Kenneth R Brown. Concatenated steane code with single-flag syndrome checks. Physical Review A, 110(3):032411, 2024. DOI: 10.1103/PhysRevA.110.032411.

[36] Armanda O Quintavalle and Earl T Campbell. Reshape: A decoder for hypergraph product codes. IEEE Transactions on Information Theory, 68(10):6569–6584, 2022. DOI: 10.1109/TIT.2022.3184108.

[37] Robert Raussendorf and Jim Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. Physical Review Letters, 98(19):190504, 2007. DOI: 10.1103/PhysRevLett.98.190504.

[38] Nithin Raveendran, Narayanan Rengaswamy, Filip Rozpędek, Ankur Raina, Liang Jiang, and Bane Vasić. Finite rate QLDPC-GKP coding scheme that surpasses

the css hamming bound. Quantum, 6:767, 2022. DOI: 10.22331/q-2022-07-20-767.

[39] Ben W Reichardt, David Aasen, Rui Chao, Alex Chernoguzov, Wim van Dam, John P Gaebler, Dan Gresh, Dominic Lucchetti, Michael Mills, Steven A Moses, et al. Demonstration of quantum computation and error correction with a tesseract code. arXiv preprint arXiv:2409.04628, 2024. DOI: 10.48550/arXiv.2409.04628.

[40] Tom Richardson. Error floors of ldpc codes. In Proceedings of the annual Allerton conference on communication control and computing, volume 41, pages 1426–1435, 2003.

[41] Joschka Roffe. LDPC: Python tools for low density parity check codes, 2022. URL https://pypi.org/project/ldpc/.

[42] Joschka Roffe, David R White, Simon Burton, and Earl Campbell. Decoding across the quantum low-density parity-check code landscape. Physical Review Research, 2(4): 043423, 2020. DOI: 10.1103/PhysRevResearch.2.043423.

[43] Kaavya Sahay, Yingjia Lin, Shilin Huang, Kenneth R Brown, and Shruti Puri. Error correction of transversal cnot gates for scalable surface-code computation. PRX quantum, 6(2):020326, 2025. DOI: 10.1103/PRXQuantum.6.020326.

[44] Thomas R Scruby, Timo Hillmann, and Joschka Roffe. High-threshold, low-overhead and single-shot decodable fault-tolerant quantum memory. arXiv preprint arXiv:2406.14445, 2024. DOI: 10.48550/arXiv.2406.14445.

[45] Luka Skoric, Dan E Browne, Kenton M Barnes, Neil I Gillespie, and Earl T Campbell. Parallel window decoding enables scalable fault tolerant quantum computation. Nature Communications, 14(1):7040, 2023. DOI: https://doi.org/10.1038/s41467-023-42482-1.

[46] Andrew Steane. Multiple-particle interference and quantum error correction. Proc. R. Soc. Lond. A., 452(1954):2551–2577, 1996. DOI: 10.1098/rspa.1996.0136.

[47] Ryan Tiew and Nikolas P Breuckmann. Low-overhead entangling gates from generalised dehn twists. IEEE Transactions on Information Theory, 71(7):5452–5468, 2025. DOI: 10.1109/TIT.2025.3571197.

[48] Jean-Pierre Tillich and Gilles Zémor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. IEEE Transactions on Information Theory, 60(2):1193–1202, 2013. DOI: 10.1109/TIT.2013.2292061.

[49] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. Physical Review A, 90(6):062320, 2014. DOI: 10.1103/PhysRevA.90.062320.

[50] Maxime A Tremblay, Nicolas Delfosse, and Michael E Beverland. Constant-overhead quantum error correction with thin planar connectivity. Physical Review Letters, 129(5):050504, 2022. DOI: 10.1103/PhysRevLett.129.050504.

[51] Yue Wu and Lin Zhong. Fusion Blossom: Fast MWPM decoders for QEC. In 2023 IEEE International Conference on Quantum Computing and Engineering (QCE), volume 1, pages 928–938. IEEE, 2023. DOI: 10.1109/QCE57702.2023.00107.

[52] Qian Xu, J Pablo Bonilla Ataides, Christopher A Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D Lukin, Liang Jiang, and Hengyun Zhou. Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. Nature Physics, 20(7):1084–1090, 2024. DOI: 10.1038/s41567-024-02479-z.

## A Syndrome extraction depth

The directions of the quantum Tanner graph's edges are determined by assigning the signs of the classical Tanner graph's edges, as explained in Sec. 4. For each vertex of the classical Tanner graph, the incident edges need to be distributed $-$ and $+$ signs as evenly as possible, such that the syndrome extraction depth of the syndrome extraction circuit is minimized [50]. We present a simple heuristic algorithm that balances the signs for each vertex below.

**Algorithm 1:** Balanced sign assignment

**Input:** List $E$ of classical Tanner graph's edges $(i, j)$ ($i \in [r]$, $j \in [n]$).
**Output:** Signs $\sigma_{ij} = -1$ or $+1$ assigned to each edge $(i, j) \in E$

**Initialization:** $\mu_i = 0$, $\nu_j = 0$ $\forall\, i \in [r]$, $j \in [n]$
**for** $(i, j) \in E$ **do**
  Generate a random number $\xi \in [0, 1)$
  **if** $\mu_i + \nu_j > 0$ **or** $\mu_i + \nu_j = 0$ **and** $\xi \geq 0.5$ **then**
    $\sigma_{ij} = +1, \quad \mu_i = \mu_i - 1, \quad \nu_j = \nu_j - 1$
  **else**
    $\sigma_{ij} = -1, \quad \mu_i = \mu_i + 1, \quad \nu_j = \nu_j + 1$
  **end**
**end**
**return** $\{\sigma_{ij}\}$

We compare the syndrome extraction depths of the syndrome extraction circuits generated by our method to those of the methods in Refs. [50, 52]. In our implementation of Ref. [50], to assign the signs of the classical Tanner graph's edges, a random permutation $\mathcal{P}$ is applied to the vertices $[n + r]$, such that $\mathcal{P}(j)$ and $\mathcal{P}(n+i)$ represent the positions of the $j$th data bit and $i$th check bit, respectively ($j \in [n]$, $i \in [r]$). Then, for each edge $(i, j)$ of the Tanner graph, $\sigma_{ij} = +1$ if $((\mathcal{P}(n + i) - \mathcal{P}(j)) \bmod (n + r)) \leq (n+r)/2$, else $\sigma_{ij} = -1$. The directions of the quantum Tanner graph's edges are assigned the same way as described in Sec. 4. Note that we assume a random permutation is applied, as an explicit method for finding a balanced ordering is not provided in Ref. [50]. We do not report the BPC code circuit's syndrome extraction depth for this method, as the positions of the classical vertices are ill-defined. Meanwhile, in Ref. [52], the $Z$-type and $X$-type syndromes are separately extracted. The syndrome extraction depth is deterministically given by $4\Delta_c$, where $\Delta_c$ is the maximum degree of the check node in the classical Tanner graph. We note that this syndrome extraction circuit is designed to suit the parallel rearrangement capabilities of the reconfigurable atom array platform [4], and a method for lowering the syndrome extraction depth is suggested in Ref. [52].

Table 1: Minimum syndrome extraction depths of the QLDPC codes used in this work obtained by various methods. The probability of obtaining the minimum depth, out of 1000 random instances, is given inside the parentheses.

| | HGP225 | HGP625 | HGP900 | QLP | BPC |
|---|---|---|---|---|---|
| M. A. Tremblay et al. [50] | 12 (36.6%) | 12 (12.9%) | 12 (7.9%) | 12 (38.3%) | - |
| Q. Xu et al. [52] | 16 | 16 | 16 | 20 | 12 |
| **This work** | **8** (1.0%) | 12 (100%) | 12 (100%) | 12 (75.4%) | **8** (100%) |

Table 1 shows the results. For all QLDPC codes considered in this work, our method either reduces the minimum syndrome extraction depth or improves the probability of obtaining the circuit with minimum syndrome extraction depth. This shows the advantage of Algorithm 1 over the previous method in which the relative positions of the bits determine the signs of the classical Tanner graph's edges.

Notably, for the $[[225, 9, 6]]$ HGP code, circuits with syndrome extraction depth 8 are found. This

achieves the lower bound on the syndrome extraction depth, which is simply the degree of the HGP code's Tanner graph [50]. A sufficient condition for meeting this lower bound is that the Tanner graphs of the two underlying classical codes consist only of vertices with even degrees and allow for a balanced ordering [50]. Our circuit shows that this condition is not a necessary condition, as each data bit is supported by 3 checks in the underlying classical code of this HGP code.

## B Circuit distance of BPC codes

The reduction in the effective distance of logical performance, as observed in Sec. 6, may occur from (i) hook errors in the syndrome extraction circuit and (ii) suboptimal decoding, due to failure of BP and/or insufficient sliding-window width. As `QUITS` offers a flexible framework for exploring different inner decoders and sliding-window parameters, here we isolate the impact of hook errors by analyzing the *circuit distance*, the minimum weight of error mechanisms in the circuit that can flip a logical observable. Note that while the circuit distance is proven to be equal to the original code distance for the HGP code [31], the reduction in circuit distance is reported for bivariate bicycle codes [6].

For various BPC codes, the circuit distance is estimated using the Stim function `search_for_undetectable_logical_errors` [17]. This function performs a heuristic search for combinations of error mechanisms that, when combined, flip a logical observable without triggering any detector. The search ends when such a combination is found. An upper bound on the circuit distance is given by the number of error mechanisms in this combination. The scripts and parameters for the search can be found in `QUITS` documentation. We choose the number of syndrome extraction rounds of the circuit such that the memory required for the search is reasonable for each BPC code; for example, finding the circuit distance for 3 rounds of $[[144, 8, 12]]$ code was infeasible with 100 GB of memory.

Table 2: Upper bounds on the circuit distances of BPC codes' syndrome extraction circuits

| BPC code parameters | Number of rounds | Circuit distance |
|---|---|---|
| $[[72, 8, 8]]$ | 3 | $\leq 6$ |
| $[[90, 8, 10]]$ | 3 | $\leq 8$ |
| $[[144, 8, 12]]$ | 2 | $\leq 9$ |

The results are presented in Table 2. The reduction of the upper bound on the circuit distances from the original code distance indicates that hook errors potentially contribute to the reduced slopes we observe for BPC codes in Fig. 5(c). Future work needs to be done on optimizing the syndrome extraction circuits such that hook errors are minimized.