Bias-tailored quantum LDPC codes

Joschka Roffe[1,2], Lawrence Z. Cohen[3], Armanda O. Quintavalle[2,4], Daryus Chandra[5], and Earl T. Campbell[2,4,6]

[1]Dahlem Center for Complex Quantum Systems, Freie Universität Berlin, 14195 Berlin, Germany

[2]Department of Physics and Astronomy, University of Sheffield, Sheffield S3 7RH, United Kingdom

[3]Centre for Engineered Quantum Systems, School of Physics, University of Sydney, Sydney, New South Wales 2006, Australia

[4]Riverlane, Cambridge CB2 3BZ, United Kingdom

[5]School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom

[6]AWS Center for Quantum Computing, Cambridge CB1 2GA, United Kingdom

Bias-tailoring allows quantum error correction codes to exploit qubit noise asymmetry. Recently, it was shown that a modified form of the surface code, the XZZX code, exhibits considerably improved performance under biased noise. In this work, we demonstrate that quantum low density parity check codes can be similarly bias-tailored. We introduce a bias-tailored lifted product code construction that provides the framework to expand bias-tailoring methods beyond the family of 2D topological codes. We present examples of bias-tailored lifted product codes based on classical quasi-cyclic codes and numerically assess their performance using a belief propagation plus ordered statistics decoder. Our Monte Carlo simulations, performed under asymmetric noise, show that bias-tailored codes achieve several orders of magnitude improvement in their error suppression relative to depolarising noise.

1 Introduction

All current quantum computing architectures are subject to the same fundamental problem: quantum bits (qubits) are extremely susceptible to error. To counter this, quantum error correction (QEC) [1, 2] must be incorporated into the quantum computing stack. With an appropriate choice of QEC code, it is in principle possible to build arbitrarily large fault tolerant quantum computers. However, the trade-off is that QEC can considerably increase the device overhead in terms of the total number of qubits required.

Quantum codes are commonly designed and benchmarked under the assumption that their qubits are subject to depolarising noise for which X (bit), Z (phase) and Y-type Pauli errors occur with equal probability. In practice, however, physical qubits are subject to biased noise where one species of error is stronger than the others. For example, in some superconducting qubit architectures phase-noise dominates by as much as five orders of magnitude [3–5]. A *bias-tailored* quantum code is designed to exploit knowledge of such noise asymmetry to improve QEC performance. Previous studies have shown that bias-tailored QEC codes, implemented in conjunction with bias-preserving two-qubit gates [6], could considerably reduce the overhead associated with achieving fault tolerant quantum computation.

In recent work by Bonilla Ataides et al. [7], it was shown that a variant of the surface code – the XZZX code – can achieve remarkable performance under biased noise. The XZZX surface code is obtained from the standard surface code via a local modification of stabilisers [8–10]. The principal advantage of the XZZX surface code is that its structure simplifies to a set of decoupled repetition codes in the infinite-bias regime. As such, the XZZX surface code is readily decodable with a threshold upper-bounded at 50% for infinite-bias. This infinite-bias threshold considerably improves on the corresponding threshold of 10.9% [11] for the standard surface code.

The surface code can be embedded onto a geometrically local architecture where gates are only performed between nearest-neighbours. This makes the surface code (and its variants) an attractive QEC scheme for the current generation of quantum architectures which typically have constrained inter-qubit connectivity [5, 12, 13]. Despite this, the surface code may not be a practical solution for fault tolerance in full-scale quantum computers. The main drawback is that it has poor rate: each surface code patch encodes only a single logical qubit. As such, quantum computers based on the surface code will require high-qubit overheads [14].

Several no-go results suggest the construction of local (or near-local) high-rate, high-distance QEC codes will not be feasible [15–17]. Fortunately, various technologies are currently in development that promise long-range inter-qubit connectivity [18–21]. On these platforms it will be possible to run *quantum low density parity check (LDPC) codes* in place of the surface code [22, 23]. Such quantum LDPC codes encode mul-

Joschka Roffe: joschka@roffe.eu, https://roffe.eu

tiple qubits per logical block and promise a route to efficient fault tolerant quantum computation.

In classical communication, LDPC codes are some of the best performing error correction protocols, underpinning technologies such as the 5G mobile standard, Gigabit Ethernet and WiFi (802.11n, 802.11ac) [24, 25]. At first, it was not known whether it would be possible to construct quantum LDPC codes with parameter scaling comparable to the best-known classical protocols. However, a series of recent advances have shown that asymptotically *good* quantum LDPC do exist [26–28]. Most recently, Pantaleev and Kalachev introduced *lifted product codes* as a family of quantum LDPC codes with finite-rate and linear block-length-to-distance scaling [28, 29]. With the underlying theoretical foundation now set, the challenge remains in engineering quantum LDPC codes into practical architectures for scalable fault tolerant quantum computing.

1.1 Summary of results

In this paper, we introduce a *bias-tailored lifted product* that enables the construction of quantum LDPC codes that exploit qubit noise asymmetry. Our bias-tailored lifted product is obtained via a local Clifford transformation of the stabilisers of the original lifted product introduced by Pantaleev and Kalachev [28]. The new stabilisers ensure that in the limit of infinite-bias, the effective distance of the quantum code is increased. This enables bias-tailored lifted product codes to harness the increased code capacity of biased error channels.

We construct explicit examples of bias-tailored lifted product codes based on classical quasi-cyclic codes [30]. These codes have complex syndrome patterns meaning they cannot be decoded using the minimum-weight perfect matching algorithms that work well for topological codes. Instead, we show that bias-tailored lifted product codes can be efficiently decoded using a belief propagation plus ordered statistics decoder (BP+OSD) [31, 32]. The decoding routine uses a method that improves performance by exploiting knowledge of correlations that arise between the bit- and phase-decoding components due to Pauli-Y errors. This procedure is outlined in detail in Appendix D, and our simulation scripts are available open source on Github [33].

To benchmark bias-tailored lifted product codes, we perform extensive BP+OSD decoding simulations across a number of bias regimes and compare to the case of depolarising noise. Our results show that the bias-tailored lifted product achieves improved logical error suppression for any biased error channel in which X- or Z-Pauli noise dominates. In the limit of infinite-bias, we observe logical error rates several orders of magnitude less than that obtained for depolarising noise. In contrast, quantum LDPC codes constructed

from the original lifted product typically suffer from a reduction in code performance with increasing noise asymmetry. We also demonstrate that quantum codes can be specifically designed to respond better to Z-bias compared to X-bias (or vice-versa).

We conjecture that the bias-tailored lifted product provides a versatile framework for the construction of codes that respond well to noise asymmetry. As an example of the construction's utility, we demonstrate that an XZZX toric code can be directly derived from the bias-tailored lifted product. Similar to the XZZX surface code, the XZZX toric code responds well to noise-asymmetry due to the fact it decouples to a set of independent repetition codes in the infinite-bias limit. Additionally, the XZZX toric code benefits from twisted periodic boundary conditions that lead to logical operators that wrap around the lattice. We show that both of these features arise automatically from the bias-tailored lifted product of two repetition codes. This demonstrates that the bias-tailored lifted product applies not just to the construction of random quantum LDPC codes, but also to structured topological codes.

This paper is structured as follows. In Section 2, we cover background in classical coding theory. This includes a description of how quasi-cyclic LDPC codes are constructed from base protographs. Section 3 outlines the essential elements of the stabiliser framework for QEC relevant to this work. In particular, we describe how quantum LDPC codes can be obtained from classical codes via product constructions. In Section 4, we present our methods for bias-tailoring QEC codes. We begin by unpicking the essential characteristics that enable XZZX codes to respond well to biased error models. We then show that the XZZX toric code can be described as a special case of a lifted product we refer to as the bias-tailored lifted product. This provides the framework through which the concept of bias tailoring can be generalised to quantum LDPC codes. Finally, in Section 5, we present the findings of our Monte Carlo decoding simulations of quantum LDPC codes under different bias regimes.

2 Classical Error Correction

2.1 Binary codes

A classical binary error correction code $\mathcal{C}$ describes a redundant mapping of a k-bit string $\mathbf{b}$ to an n-bit codeword $\mathbf{c}$, where $n > k$ [1]. The codewords $\mathbf{c} \in \mathcal{C} = \text{NULLSPACE}(H)$ are the null-space vectors of an $m \times n$ binary parity check matrix H such that $H \cdot \mathbf{c} \mod 2 = \mathbf{0}$ (note that from this point onward,

---

[1]We use bold symbols to denote binary strings. For example, $\mathbf{b} = 01$. When involved in matrix operations we assume these binary strings are equivalent to column vectors. Eg. $\mathbf{b} = 01 = [0, 1]^T$.
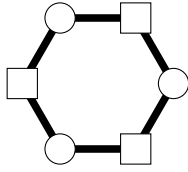
Figure 1: The factor graph of the distance-three closed loop repetition code. The circular nodes represent data bits whilst the square nodes represent parity bits. Edges are drawn between nodes according to the parity check matrix H_3^{rep} defined in Eq. (1).

we will assume mod 2 arithmetic for operations involving bit strings and parity check matrices). By the rank-nullity theorem, the number of bits k encoded by code $\mathcal{C}$ is related to the rank of the parity check matrix H such that $k = n - \mathrm{rank}(H)$. The code distance d is defined as the minimum Hamming weight $d = \min(\mathrm{wt}(\mathbf{c}))$ of a non-zero codeword $\mathbf{c} \in \mathcal{C}$. If a codeword $\mathbf{c} \in \mathcal{C}$ is subject to an error $\mathbf{e}$, this error can be detected by computing the code syndrome $\mathbf{s} = H \cdot (\mathbf{c} + \mathbf{e}) = H \cdot \mathbf{e}$. Code syndromes are non-zero for all errors $\mathbf{e}$ of Hamming weight less than the code distance $\mathrm{wt}(\mathbf{e}) < d$. We adopt the convention of labelling classical codes in terms of the $[n, k, d]$ notation, where n is the block length (codeword length), k is the dimension (number of encoded bits) and d is the code distance. Another important figure of merit is the code rate r, defined as the ratio of encoded bits to the block length $r = k/n$.

Example 2.1. The parity check matrix for a length-three closed-loop repetition code is given below

$$H_3^{\mathrm{rep}} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \tag{1}$$

The codewords of H_3^{rep} are given by $\mathcal{C} = \mathrm{NULLSPACE}(H_3^{\mathrm{rep}}) = \{000, 111\}$. The block length of this code is $n = 3$. By the rank-nullity theorem, the number of encoded bits is $k = n - \mathrm{rank}(H) = 1$. The lowest-weight non-zero codeword is 111 giving a code distance of $d = 3$. The $[n, k, d]$ code parameters of H_3^{rep} are therefore $[3, 1, 3]$. As an example of error detection, the error $\mathbf{e} = 100$ would give a syndrome $\mathbf{s} = H \cdot \mathbf{e} = 101$. The distance of a repetition code can be improved simply by increasing the length of the codewords (increasing the number of repetitions). In general, a length-n repetition code will have code parameters $[n, k, d] = [n, 1, n]$.

2.2 Factor graphs

The factor graph of a code is a bipartite graph with an adjacency matrix given by the parity check matrix H: the columns of H (data bits) map to the round nodes and the rows of H (check bits) map to

the square nodes. As an example, Fig. 1 shows a factor graph representation of the length-three repetition code. The corresponding adjacency matrix is given by H_3^{rep} defined in Eq. (1). From the factor graph, we see that larger repetition codes can be obtained simply by increasing the length of the loop. In addition to providing a useful visualisation of the code, factor graphs serve a purpose in code design, analysis and decoding [32, 34].

2.3 Quasi-cyclic LDPC codes

Low density parity check (LDPC) codes are classical binary codes with sparse factor graphs. More concisely, a code is labelled (l, q)-LDPC when the column and row weights of its parity check matrix are upper bounded by l and q respectively. There are many methods for constructing high-performance LDPC codes for a variety of use cases. In the setting of classical communication, one of the most widely used methods is the quasi-cyclic construction [30].

The fundamental building blocks of quasi-cyclic codes are circulant permutation matrices. We define a circulant permutation matrix λ_L^t as the matrix obtained by shifting the columns of an $L \times L$ identity matrix $\mathbb{1}_L$ by t positions to the right. As an example, the 3×3 circulant permutation matrix with shift parameter $t = 1$ is given by

$$\lambda_3^1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}. \tag{2}$$

It is clear that after L shifts, the circulant permutation matrix λ_L^L is equal to the identity $\lambda_L^L = \lambda_L^0 = \mathbb{1}_L$. In general, circulant permutation matrices obey the cyclic property

$$\lambda_L^t = \lambda_L^{t \bmod L}. \tag{3}$$

The set $\{\lambda_L^0, \lambda_L^1, ..., \lambda_L^{L-1}\}$ of L distinct permutation matrices – including the identity – forms a basis of a vector space $\mathbb{A}_L$ over the binary field $\mathbb{F}_2$. Any element $\alpha \in \mathbb{A}_L$ can be written as a linear combination of the elements from this basis:

$$\alpha = \sum_{l=0}^{L-1} a_l \lambda_L^l, \tag{4}$$

where $a_l \in \mathbb{F}_2$. The product of any two circulant permutation matrices is also a permutation matrix, and it holds:

$$\lambda_L^i \cdot \lambda_L^j = \lambda_L^{i+j \bmod L}. \tag{5}$$

In general,

$$\alpha \cdot \beta \in \mathbb{A}_L, \tag{6}$$

for any pair of elements $\alpha, \beta \in \mathbb{A}_L$. Hence, $\mathbb{A}_L$ is a *ring* and we refer to it as the *ring of circulants*[2].

The binary codes introduced in Section 2.1 have parity check matrices with coefficients in $\mathbb{F}_2$. To construct a quasi-cyclic code, we first define an $m' \times n'$ matrix $A \in \mathbb{A}_L^{m' \times n'}$ with coefficients in the ring of circulants $\mathbb{A}_L$. This matrix is sometimes referred to as a *protograph* and provides a compact means of representing a large quasi-cyclic code in terms of its constituent permutation matrices. We indicate by $\mathfrak{B}(.)$ the mapping that transforms a protograph to its binary representation

$$\mathfrak{B}(.): \quad A \in \mathbb{A}_L^{m' \times n'} \mapsto \mathfrak{B}(A) \in \mathbb{F}_2^{m \times n}. \quad (7)$$

The binary matrix $\mathfrak{B}(A)$ is obtained from A by replacing each coefficient $A_{ij} = \sum_{l=0}^{L-1} \alpha_l \lambda_L^l$ of A with its binary representation, i.e. by substituting each λ_L^l with the corresponding permutation matrix in $\mathbb{F}_2^{L \times L}$. In general, an $m' \times n'$ protograph $A \in \mathbb{A}_L^{m' \times n'}$ yields an $m \times n$ binary matrix $\mathfrak{B}(A) \in \mathbb{F}_2^{m \times n}$ where $m = m'L$ and $n = n'L$.

We define the *weight* of $\alpha \in \mathbb{A}_L$ as the number of non zero terms in the decomposition defined in Eq. (4), i.e. if $\alpha = a_0 \mathbb{1} + \cdots + a_{L-1}\lambda_L^{L-1}$, $\text{wt}(\alpha) = |\{i \text{ s.t } a_i = 1\}|$. For example, $\text{wt}(\lambda_0^L) = 1$ and $\text{wt}(\lambda_0^0 + \lambda_L^1) = 2$. We associate to each protograph $A \in \mathbb{A}_L^{m' \times n'}$, a *weight enumerator matrix* $W \in \mathbb{Z}^{m' \times n'}$ such that $W_{ij} = \text{wt}(A_{ij})$. Since each permutation matrix has rows and columns of weight one, it is easy to verify that the row/column weight of W equates the row/column weight of the binary matrix $\mathfrak{B}(A)$. This equivalence allows us to bound the sparsity of the quasi-cyclic LDPC code with parity check matrix $\mathfrak{B}(A)$ in terms of the protograph A. A protograph whose weight enumerator matrix has row and column weights upper-bounded by l and q respectively yields an (l, q)-LDPC code.

Example 2.2. Consider the quasi-cyclic code defined by the following 2×3 protograph

$$A_L = \begin{bmatrix} \lambda_L^1 + \lambda_L^2 & \lambda_L^0 & 0 \\ 0 & \lambda_L^0 + \lambda_L^1 & \lambda_L^1 \end{bmatrix} \quad (8)$$

The weight enumerator matrix of A_L is given by

$$W = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}. \quad (9)$$

The column and row weights of W are upper bounded by three and three respectively. Any parity check matrix $\mathfrak{B}(A_L)$ derived from the protograph A_L will

therefore be (3,3)-LDPC. When $L = 3$, the protograph A_3 yields a $[9, 3, 3]$ code with the following parity check matrix

$$\mathfrak{B}(A_3) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}. \quad (10)$$

Example 2.3. The family of distance-L closed-loop repetition codes can be interpreted as quasi-cyclic codes defined by the following 1×1 protograph

$$A_L^{\text{rep}} = \left[\lambda_L^0 + \lambda_L^1 \right]. \quad (11)$$

The weight enumerator matrix of A_L^{rep} is $W = [2]$. As such, for any $L > 1$, the repetition code with parity check matrix $\mathfrak{B}(A_L^{\text{rep}})$ is (2,2)-LDPC. It is clear that, for $L = 3$, $\mathfrak{B}(A_3^{\text{rep}})$ is equal to the parity check matrix of the distance-three repetition code defined in Eq. (1).

3 Quantum error correction

3.1 Stabiliser codes

Similar to classical binary codes, a stabiliser QEC code describes a redundant mapping from a K-qubit state $|\psi\rangle$ to an N-qubit logical state $|\psi\rangle_L$. Qubit errors are described by linear combinations of matrices from the basis $\{\mathbb{1}, X, Y, Z\} \in \mathcal{G}$, where $\mathcal{G}$ is the Pauli group[3]. The logical states are defined so that they project onto the $(+1)$ eigenspace of a group of Pauli operators $\mathcal{S} \subset \mathcal{G}_N$ known as the code stabilisers such that $S |\psi\rangle_L = (+1) |\psi\rangle_L$ for all $S \in \mathcal{S}$. If the logical state is subject to a Pauli error E that anti-commutes with a stabiliser $S \in \mathcal{S}$, then the measurement of this stabiliser projects onto the (-1) eigenspace $S_j E |\psi\rangle_L = (-1) |\psi\rangle_L$. The results of the stabiliser measurements form a syndrome that can be decoded to determine the best recovery operation.

Stabiliser codes with dimension $K > 0$ have a set of logical operators that allow computation directly on the encoded states. The logical operators are defined as operators that commute with all of the code stabilisers, but are themselves not stabilisers. The encoded quantum information is lost when the code is subject to a Pauli error that is equivalent to a non-trivial logical operator. The minimum weight of a logical operator therefore gives the distance D of a quantum code. In general, QEC codes are labelled in

---

[2]A ring $\mathbb{A}$ is a set equipped with addition and multiplication such that it is a group under addition and it is closed under multiplication. Since permutation matrices λ_L^t satisfy the polynomial equation $x^L - 1 = 0$, $\mathbb{A}_L$ can be expressed as a quotient ring $\mathbb{F}_2[x]/(x^L - 1)$. For more details, see, for instance, [35].

[3]The single-qubit Pauli group $\mathcal{G}_1 = \langle I, X, Y, Z \rangle$ where $\mathbb{1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ and $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$. The Pauli-group over N-qubits is given by $\mathcal{G}_N = \bigotimes_{i=0}^N \mathcal{G}_i$.

terms of an $[[N, K, D]]$ notation, where the N is total number of qubits, K is the number of encoded logical qubits and D is the code distance. Note the use of double brackets to differentiate quantum codes from classical codes.

There is a useful mapping for Pauli operators that allows stabiliser QEC codes to be expressed in terms of a binary representation. For single Pauli operators this mapping acts as follows: $\mathbb{1} \mapsto (0|0)$, $X \mapsto (1|0)$, $Y \mapsto (1|1)$ and $Z \mapsto (0|1)$. In general, an n-qubit Pauli operator G can be mapped to $2N$-bit binary vector of the form

$$G \mapsto \mathbf{g} = (\mathbf{g}_x | \mathbf{g}_z) \in \mathbb{F}_2^{2N}, \qquad (12)$$

where $\mathbf{g}_x$ is a n-bit binary vector set to 1 in the qubit locations where G acts with an X-operator. Similarly, $\mathbf{g}_z$ marks the qubit locations where G acts with a Z operator. A Y-Pauli operator acting on qubit i results in a 1 in the corresponding position of both vectors $\mathbf{g}_x$ and $\mathbf{g}_z$. As an example, the two-qubit operator $G = X_1 Y_2$ maps to the binary representation $\mathbf{g} = (11|01)$.

Using the binary representation, a QEC code can be written in terms of an $M \times 2N$ parity matrix, $H_Q = [H_X | H_Z] \in \mathbb{F}_2^{M \times 2N}$, where each row corresponds to one of the M stabiliser generators. For an error $\mathbf{e}_Q = (\mathbf{e}_X | \mathbf{e}_Z)$ the code syndrome is given by

$$\mathbf{s}_Q = [H_X | H_Z] \cdot (\mathbf{e}_Z | \mathbf{e}_X) = H_X \cdot \mathbf{e}_Z + H_Z \cdot \mathbf{e}_X. \quad (13)$$

In the following sections we will see the binary representation of stabiliser codes facilitates the re-purposing of existing classical codes into quantum codes. Another useful feature of the binary representation is that it enables the straightforward verification of various code properties. For example, the condition that code stabilisers commute amounts to verifying that H_X and H_Z satisfy the following constraint

$$H_X \cdot H_Z^T + H_Z \cdot H_X^T = 0. \qquad (14)$$

3.2 Calderbank, Shor and Steane (CSS) codes

The Calderbank, Shor and Steane (CSS) family is a subset of QEC codes that have disjoint X- and Z-type stabilisers such that each stabiliser is made up either exclusively of X-type Pauli operators, or exclusively of Z-type Pauli operators [36–38]. In the binary representation, the parity matrix of a CSS code has the form

$$H_{\text{CSS}} = \begin{bmatrix} 0 & H_Z' \\ H_X' & 0 \end{bmatrix}. \qquad (15)$$

For CSS codes, the stabiliser commutativity constraint defined in Eq. (14) simplifies to verifying that $H_X' \cdot H_Z'^T = 0$. The structure of CSS codes allows bit and phase-flips to be corrected using separate classical codes. However, the commutativity constraint on stabiliser codes means that it is not simply possible to

re-purpose any classical code as the H_X' and H_Z' matrices of a CSS code. Various constructions exist that allow classical codes to be converted to CSS codes. In this work, we focus on two such constructions: the hypergraph product [39] and the lifted product [28, 31].

3.3 Hypergraph product codes

The hypergraph product is a method for creating a quantum code from any two classical *seed* codes [39, 40]. To construct a hypergraph product code, we first choose the seed codes: $H_1 \in \mathbb{F}_2^{m_1 \times n_1}$ with parameters $[n_1, k_1, d_1]$ and $H_2 \in \mathbb{F}_2^{m_2 \times n_2}$ with parameters $[n_2, k_2, d_2]$. For each seed code $H \in \mathbb{F}_2^{m \times n}$, we also define a transpose code with parity check matrix $H^T \in \mathbb{F}_2^{n \times m}$ and parameters $[m, k^T, d^T]$. Given seed codes H_1 and H_2, the parity check matrix of the hypergraph product code is as follows

$$H_{HGP} = [H_X | H_Z]$$
$$= \begin{bmatrix} 0 & 0 & \mathbb{1}_{n_1} \otimes H_2 & H_1^T \otimes \mathbb{1}_{m_2} \\ H_1 \otimes \mathbb{1}_{n_2} & \mathbb{1}_{m_1} \otimes H_2^T & 0 & 0 \end{bmatrix}. \quad (16)$$

The above code H_{HGP} has parameters $[[N, K, D]]$ where $N = n_1 n_2 + m_1 m_2$, $K = k_1 k_2 + k_1^T k_2^T$ and $D = \min(d_1, d_2, d_1^T, d_2^T)$. The specific advantage of the hypergraph product is that any pair of classical codes can be used as the seed matrices: it is straightforward to verify that the stabiliser commutivity constraint holds for all pairs H_1 and H_2. Another useful property is that the hypergraph product preserves the sparsity of the seed codes (up to a constant additive factor). The hypergraph product of two LDPC codes therefore yields a quantum code with an LDPC parity check matrix.

Both the H_X and H_Z components of the matrix in Eq. (16) are formed from a concatenation of two blocks of qubits. We refer to the first block of $n_1 n_2$ qubits as *sector one* and the second block of $m_1 m_2$ qubits as *sector two*.

Example 3.1. The CSS toric code is constructed from the hypergraph product by choosing closed-loop repetition codes as the seed codes H_1 and H_2. Fig. 2 shows the factor graph of a toric code constructed from the hypergraph product of a length-three repetition code ($H_1 = H_3^{\text{rep}}$) and a length-two repetition code ($H_2 = H_2^{\text{rep}}$). Here, the circular nodes represent the data qubits and the square nodes the check qubits which measure the stabilisers. The solid edges denote Z-type Pauli checks and are drawn according to the adjacency matrix H_Z, whilst the dashed edges represent X-type checks and are drawn according to H_X. Boundary edges are labelled to indicate which qubit they connect to on the opposite side of the lattice. We see that each column of the toric code contains a copy of the seed code H_1, whilst each row contains a copy of seed code H_2. By convention, we define a toric code lattice to have dimension $n_1 \times n_2$. The
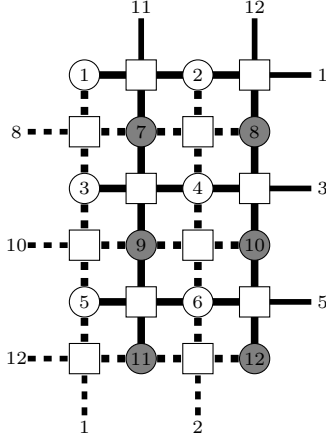
Figure 2: A $[[12, 2, 2]]$ CSS toric code obtained from the hypergraph product of a $d = 3$ repetition code ($H_1 = H_{\text{rep}}^3$) with a $d = 2$ repetition code ($H_2 = H_{\text{rep}}^2$). The circular nodes represent data qubits and the square nodes the check qubits. Solid edges denote Z-type Pauli checks and dashed edges X-checks. The boundary edges are labelled to indicate which qubits they connect to on the opposite side of the lattice. The subset of data qubits filled with white are in qubit sector 1 (corresponding to the first block of $n_1 n_2$ qubits in the hypergraph product) whilst the subset filled in grey are the sector two qubits (corresponding to the second block of $m_1 m_2$ qubits in the hypergraph product).

depicted code has parameters $[[12, 2, 2]]$ defined on a 3×2 lattice. Repetition codes have square parity check matrices which means that $n_{1,2} = m_{1,2}$ for the seed codes H_1 and H_2. Consequently, toric codes have an equal number of qubits in both sector one and sector two. In Fig 2, the sector one qubits are labelled one to six (filled in white) and the sector two qubits six to twelve (filled in grey). Bigger toric codes can be constructed by using larger repetition codes as the hypergraph seeds.

Example 3.2. The parity check matrix H given by Eq. (37) in Appendix A defines a classical code with parameters $[16, 4, 6]$. Choosing $H_1 = H$ and $H_2 = H$, we obtain a hypergraph product code with parameters $[[400, 16, 6]]$.

3.4 Lifted product codes

The hypergraph product construction described by Eq. (16) is applied to a pair of seeds codes defined by binary parity check matrices. The lifted product applies the same operation, but to a pair of seed protographs, which are matrices with coefficients in the ring of circulants $\mathbb{A}_L$ as defined in Section 2.3. Before introducing the lifted product, we describe the transpose operation for elements in $\mathbb{A}_L$. For an element $\alpha = \sum_{i=0}^{L-1} a_i \lambda_L^i$, we define its transpose α^T as

$$\alpha^T := \sum_{i=0}^{L-1} a_i \lambda_L^{(-i)} = \sum_{i=0}^{L-1} a_i \lambda_L^{(L-i)}. \qquad (17)$$

For example, if $\alpha = \lambda_L^0 + \lambda_L^1$ then $\alpha^T = \lambda_L^0 + \lambda_L^{(-1)}$. Given an $m \times n$ protograph $A \in \mathbb{A}_L^{m \times n}$ with coefficients $A_{ij} \in \mathbb{A}_L$, the transpose is an $n \times m$ protograph $A^T \in \mathbb{A}_L^{n \times m}$ with coefficients $(A^T)_{ij} = (A_{ji})^T$. We also define an identity protograph, $E_m \in \mathbb{A}_L^{m \times m}$, as an $m \times m$ diagonal matrix where all non-zero entries are $\lambda_L^0 \in \mathbb{A}_L$ such that

$$E_m = \text{diag}(\lambda_L^0, \ldots, \lambda_L^0) \in \mathbb{A}_L^{m \times m}. \qquad (18)$$

Mapped to binary, the identity protograph is an size-m^2 identity matrix: $\mathcal{B}(E_m) = \mathbb{1}_{m^2}$. For example, the identity protograph $E_3 \in \mathbb{A}_L^{3 \times 3}$ is given by

$$E_3 = \begin{bmatrix} \lambda_3^0 & 0 & 0 \\ 0 & \lambda_3^0 & 0 \\ 0 & 0 & \lambda_3^0 \end{bmatrix}, \qquad (19)$$

which maps to a size-9 identity matrix $\mathbb{1}_9$. The identity protograph $E_m \in \mathbb{A}_L^{m \times m}$ leaves all protographs $A \in \mathbb{A}_L^{m \times n}$ unchanged under matrix multiplication $E_m \cdot A = A$. Finally, we note that the tensor product between two protographs is well defined as a simple extension of the standard tensor product to the ring of circulants: the tensor product between $A \in \mathbb{A}_L^{m \times n}$ and $B \in \mathbb{A}_L^{l \times k}$ yields the protograph $C = A \otimes B \in \mathbb{A}_L^{ml \times nk}$. As an example, the tensor product between $E_2 \in \mathbb{A}_L^{2 \times 2}$ and $A \in \mathbb{A}_L^{m \times n}$ gives the following block protograph

$$E_2 \otimes A = \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix} \in \mathbb{A}_L^{2m \times 2n}. \qquad (20)$$

To construct a lifted product code, we first choose two protographs: $A_1 \in \mathbb{A}_L^{m_1' \times n_1'}$ with dimensions $m_1' \times n_1'$ and $A_2 \in \mathbb{A}_L^{m_2' \times n_2'}$ with dimensions $m_2' \times n_2'$. Given these protographs, the lifted product defines a quantum protograph A_Q

$$A_Q = [A_X | A_Z]$$
$$= \begin{bmatrix} 0 & 0 & E_{n_1'} \otimes A_2 & (A_1)^T \otimes E_{m_2'} \\ A_1 \otimes E_{n_2'} & E_{m_1'} \otimes (A_2)^T & 0 & 0 \end{bmatrix}. \qquad (21)$$

Note that this has the same form as the hypergraph product given by Eq. (16). The CSS parity check matrix corresponding to A_Q is given by

$$\mathcal{B}(A_Q), \qquad (22)$$

where $\mathcal{B}(.)$ is the mapping defined in Eq. (7) that returns the binary representation of a protograph. The length of the lifted product code is given by $N = L(m_1' m_2' + n_1' n_2')$.

Despite similarities to the hypergraph product, the lifted product is different to the hypergraph product of $\mathcal{B}(A_1)$ and $\mathcal{B}(A_2)$, which would have length $N = L^2(m_1' m_2' + n_1' n_2')$. The crucial difference is that the lifted product applies $\mathcal{B}(.)$ after forming block matrix of Eq. (21) rather than before.

Example 3.3. Consider the following protograph

$$A_L = \begin{bmatrix} \lambda_L^0 & \lambda_L^{11} & \lambda_L^7 & \lambda_L^{12} \\ \lambda_L^1 & \lambda_L^8 & \lambda_L^1 & \lambda_L^8 \\ \lambda_L^{11} & \lambda_L^0 & \lambda_L^4 & \lambda_L^8 \\ \lambda_L^6 & \lambda_L^2 & \lambda_L^4 & \lambda_L^{12} \end{bmatrix}. \qquad (23)$$

When $L = 13$, $\mathfrak{B}(A_{13})$ is a $(4,4)$-LDPC matrix which yields a code with parameters $[52, 3, 26]$. We generated this protograph using the progressive edge growth (PEG) technique for quasi-cyclic codes [41]. The PEG method proceeds by randomly generating protograph elements under the constraint that each new term does not decrease the code girth below a certain target value. Here, the code girth is defined as the minimum-length loop in the code factor graph. Factor graphs with high-girth are beneficial as small-length loops can cause decoders based on belief propagation to fail. More specific details concerning decoding methods are discussed in Section 5. The parity check matrix $\mathfrak{B}(A_{13})$ has minimum-girth six. The code distance $d = 26$ was determined via an exhaustive search over all codewords.

By applying the lifted product to two copies of A_{13}, we obtain a quantum code with parameters $[[416, 18, d \leq 20]]$. There is no known method of computing the code distance of a generic lifted product code given the distances of the seed codes. However, we can obtain a heuristic indication of the code performance by running simulations. The upper-bound $d \leq 20$ on the code distance is an estimate based on the minimum-weight logical operator observed after extensive decoding simulation with the BP+OSD decoder. By comparison, the $[[400, 16, 6]]$ hypergraph product code from Example 3.2 has a similar code rate but a minimum distance of six.

4 Bias-tailored QEC

4.1 Error model

We now explore how QEC codes can be designed to exploit noise asymmetry. To characterise the qubit bias, we consider an error model in which each qubit is independently subject to a Pauli channel $\mathcal{E}_Q(\rho)$ of the form

$$\mathcal{E}_Q(\rho) = (1-p)\rho + p_X X\rho X + p_Y Y\rho Y + p_Z Z\rho Z, \quad (24)$$

where ρ is the density matrix of the single-qubit state and p_X, p_Y and p_Z are the physical error rates for X-, Y- and Z-type errors respectively. The total error rate p is given by the sum over the three error types $p = p_X + p_Y + p_Z$. Under this channel, the noise bias η_i is defined

$$\eta_i = \frac{p_i}{\sum_{j \neq i} p_j}, \qquad (25)$$

where $p_i, p_j \in \{p_X, p_Y, p_Z\}$. For example, the Z-error bias is $\eta_Z = 0.5$ for depolarising noise where
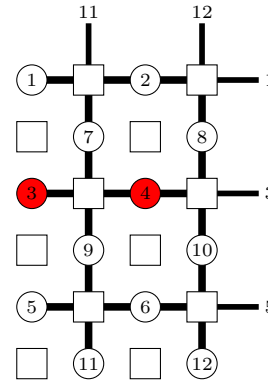

Figure 3: The H_Z component of the $[[12, 2, 2]]$ CSS toric code depicted in Fig. 2. The nodes filled with red show a weight-two logical operator that spans the lattice.

$p_X = p_Y = p_Z$. Now consider the case of infinite Z-bias $\eta_Z = \infty$ where the Z-error probability has some non-zero value $0 < p_Z \leq 1$ whilst the X- and Y-error probabilities are zero $p_X = p_Y = 0$. In this regime, the quantum syndrome equation defined in Eq.(13) simplifies to $\mathbf{s}_Q = H_X \cdot \mathbf{e}_Z$, as the probability of obtaining either an X- or Y-error is zero. The performance of the code therefore depends exclusively on the H_X component of the quantum parity check matrix. Similarly, for infinite X-bias $\eta_X = \infty$, the syndrome equation simplifies to $\mathbf{s}_Q = H_Z \cdot \mathbf{e}_X$ leaving the code performance dependent on H_Z. From this, we see that tailoring a QEC code to a specific bias-type involves ensuring that the relevant sub-component of the quantum parity check matrix (H_X or H_Z) is itself a *good* code, both in terms of its distance and the ease of its decoding.

How do we expect the capacity of the quantum error channel to change with increasing bias? A partial answer to this question can be obtained by applying the Shannon noisy coding theorem to obtain a quantity known as the *Hashing threshold*. The Hashing threshold gives a lower bound[4] on the physical error rate below which a quantum code can in theory arbitrarily suppress the error rate. For the quantum channel defined in Eq. (24), the Hashing threshold for depolarising noise is 18.9%. As the bias is increased (for X-,Y- or Z-noise), the Hashing threshold increases rapidly until converging to a value of 50% in the limit of infinite-bias. The Hashing threshold provides the theoretical motivation for bias-tailoring: the higher the bias, the higher the channel capacity. A more detailed discussion of the Hashing threshold can be found in Appendix C.

---

[4]The Hashing threshold is a lower bound as it does not account for quantum effects such as superposition and entanglement that increase the code threshold.

4.2 The CSS toric code under biased noise

We first consider the behaviour of the standard CSS code (as presented in Example 3.1) in different bias regimes. Under depolarising noise the upper bound on the threshold of the CSS toric code is 18.9% as determined by mapping the code Hamiltonian to the classically disordered eight-bond Ising model [11]. Thresholds close to this upper bound have been numerically observed via Monte Carlo simulations of the toric code using decoders based on minimum-weight perfect matching [42, 43]. From this we see that under depolarising noise, the toric code matches the Hashing threshold of 18.9%. However, as the bias is increased, the toric code's threshold begins to drop counter to what we would expect from the Hashing threshold. To explore why this occurs, we consider the infinite X-bias regime where the performance of the toric code relies exclusively on the measurement and decoding of its Z-stabilisers. Fig. 3 shows the factor graph of the H_Z subcomponent of the $[[12, 2, 2]]$ toric code introduced in Example 3.1. The nodes shaded in red depict a weight-two X-logical operator that spans the lattice. In the limit of infinite X-bias, this toric code therefore has distance $d_X = 2$. By again mapping to a statistical physics model, the threshold of H_Z (and H_X) component of the toric code under infinite-bias is upper bounded at 10.9% [11]. This is considerably lower than the Hashing threshold of 50% in the infinite-bias regime. It is here that we see the weakness of the CSS toric code when it comes asymmetric channels: its H_Z and H_X sub-components are themselves not good classical codes. Consequently, the CSS toric code is unable to leverage the increase in channel capacity that comes with increased bias.

4.3 The XZZX toric code

The XZZX surface code is a modified version of the CSS surface code that has recently been shown to perform exceptionally well under biased-noise [7, 8, 10]. The XZZX surface code, as presented in [7, 8, 10], is defined on a rotated lattice with periodic boundary conditions. The improved performance of the XZZX surface code relative to the standard CSS code can be attributed to two factors: 1) a redefinition of the stabilisers that results in H_X and H_Z sub-components reducing to a set of decoupled repetition codes with a threshold of 50%; 2) *twisted* periodic boundary conditions that lead to longer logical operators in the infinite-bias limit. In this paper, we explore an XZZX protocol based on the CSS toric code and defined on a rectangular lattice. In this setting, we will see that both XZZX modifications – stabiliser redefinition and boundary twists – can be directly derived from the lifted product.

To construct a XZZX code, the first step is to redefine the stabilisers of the hypergraph product via an operation we refer to as a *Hadamard rotation*. The
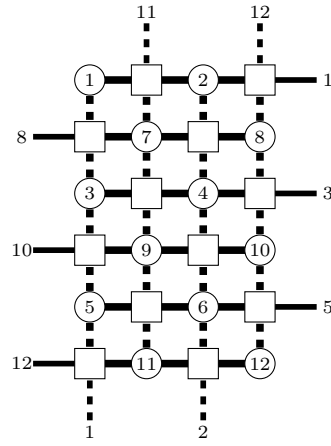

Figure 4: The $[[12, 2, 2]]$ XZZX toric code obtained after Hadamard rotating the CSS toric code. This code is non-CSS where each check measures the same XZZX stabiliser on its adjacent qubits.

Hadmard rotation applies a Hadamard gate to all sector two qubits. The effect of this is that all X-check edges incident on sector two qubits are transformed to Z-check edges and vice-versa. Following the Hadmard rotation, the hypergraph product defined in Eq. (16) has the form

$$H'_{HGP} = [H_X | H_Z]$$
$$= \begin{bmatrix} 0 & H_1^T \otimes \mathbb{1}_{m_2} & \mathbb{1}_{n_1} \otimes H_2 & 0 \\ H_1 \otimes \mathbb{1}_{n_2} & 0 & 0 & \mathbb{1}_{m_1} \otimes H_2^T \end{bmatrix}. \quad (26)$$

The above parity check matrix has mixed-type stabilisers, and is therefore no longer a CSS code. Fig. 4 shows the factor graph of the XZZX toric code obtained by applying a Hadamard rotation to the $[[12, 2, 2]]$ CSS toric code depicted in Fig. 2. The key difference is that all Z-check edges now act horizontally whilst all X-check edges act vertically. Every parity check qubit measures the same XZZX stabiliser which lends the code its name.

In addition to transforming the stabilisers, the Hadmard rotation also transforms the logical operators. For any CSS hypergraph product code, a basis of logical operators can be written in the form

$$L_{CSS} = \begin{bmatrix} 0 & 0 & L_{Z1} & L_{Z2} \\ L_{X1} & L_{X2} & 0 & 0 \end{bmatrix}, \quad (27)$$

where we again separate the operators into the components that act on the sector one qubits and sector two qubits. In addition to transforming the code stabilisers, the Hadamard rotation also transforms the logical operators so that they read

$$L_{XZZX} = \begin{bmatrix} 0 & L_{Z2} & L_{Z1} & 0 \\ L_{X1} & 0 & 0 & L_{X2} \end{bmatrix}. \quad (28)$$

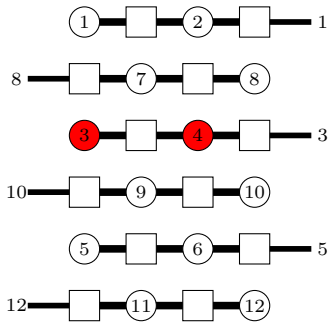From the above, it follows that the Hadamard rotation does not alter the weight of any logical operator

Figure 5: The H_Z sub-component of the $[[12, 2, 2]]$ XZZX toric code depicted in Fig 5. The Hadamard rotation ensures that the H_Z component decouples to a set of closed-loop repetition codes.

and leaves the code parameters unchanged. Consequently, we expect the depolarising noise threshold of the XZZX code to be the same as that for the CSS toric code. However, under biased noise, the performance of the XZZX toric code immediately begins to outperform that of the CSS toric code. To understand this, we again consider the case of infinite X-bias, $\eta_X = \infty$, where the performance of the code is dependent only on the sub-component H_Z. This sub-component is depicted for the $[[12, 2, 2]]$ toric code in Fig. 5, where we see that it now consists of a set of decoupled closed-loop repetition codes. The threshold of the repetition code is upper-bounded at 50%. As a result, the upper-bound on the threshold of the XZZX toric code grows from 18.9% to 50% with increasing noise bias. Herein lies the principal benefit of the XZZX construction: the Hadamard rotation modifies the stabilisers so that the H_X and H_Z components are built from copies of the seed codes (and their transposes). Therefore, provided the seeds codes are themselves *good* classical codes, the resultant XZZX hypergraph product will also be a good code in the limit of infinite-bias. In the case of the XZZX code, the seeds are repetition codes and the resultant infinite-bias threshold saturates the Hashing bound.

4.4 The XZZX toric code with twisted boundary conditions

In the infinite X-bias regime, the $[[12, 2, 2]]$ XZZX toric code has logical-X distance $d_X = 2$. The nodes filled in red Fig. 5 show an example of a minimum-weight logical operator that spans the lattice. One way of improving the infinite-bias distance d_X is to increase the width of the lattice: this amounts to using a longer repetition code as the seed H_2. The disadvantage of this method is that it results in more qubits and a reduction in the code rate. An alternative, more qubit-efficient technique for improving d_X, is to modify the code via a *boundary twist*. In the CSS toric

code derived from the hypergraph product, boundary checks connect to qubits on the opposite side of the lattice. Fig. 6 shows an XZZX toric code where the boundary conditions for the H_Z component have been *twisted* so that each boundary check connects instead to the qubit shifted one row below on the opposite side of the lattice. The resultant code has an improved distance of $d = 3$ giving it parameters $[[12, 2, 3]]$. In the infinite-bias regime, the distance is increased even further. Fig. 7 shows a logical operator (red nodes) on the H_Z sub-component of the twisted code. We see that this logical operator has weight six and covers all the qubits in sector one. Similarly, a second independent logical operator can be defined that covers all sector two qubits. In the limit of infinite X-bias, the X-distance is therefore $d_X = 6$.

In general, we define an XZZX twisted toric code on an $n_1 \times n_2$ lattice where n_1 corresponds to the number of data qubits spanning the lattice vertically and n_2 the number of data qubits spanning the lattice horizontally. Eg. the lattice depicted in Fig. 6 has lattice parameters $n_1 = 3$ and $n_2 = 2$. The parity check matrix of the XZZX twisted toric code is written

$$
\begin{aligned}
H &= [H_X | H_Z] \\
&= \begin{bmatrix} 0 & H_1^T \otimes \mathbb{1}_{n_2} & H_{N/2}^{\text{rep}} & 0 \\ H_1 \otimes \mathbb{1}_{n_2} & 0 & 0 & (H_{N/2}^{\text{rep}})^T \end{bmatrix},
\end{aligned}
\tag{29}
$$

where N is the qubit block length $N = 2n_1 n_2$. We see that the above differs from Eq. (26) in that the H_Z component has been rewritten in terms of two repetition codes of length $N/2$. The first repetition codes wraps around the sector one qubits and the second around the sector two qubits. Note that for the example in this section, we have twisted only the H_Z component. An equivalent XZZX code could be constructed by instead twisting the H_X component.

In the CSS setting, boundary twists have previously been used to improve code performance by Kovalev and Pryadko [10]. Boundary twists have also served a role in the recent discoveries of high-performance quantum LDPC codes [26, 27].

4.5 Lifted product representation of the XZZX toric code

Following the boundary twist, the parity check matrix of the XZZX twisted toric code given by Eq. (29) can no longer be derived from the hypergraph product. In this section, we show that the XZZX twisted toric code is in fact an instance of a lifted product code.

In order to recast the XZZX twisted code as a lifted product we need to find a protograph representation of the parity check matrix in Eq. (29). From Example 2.3, we note that the parity check matrix of the repetition code $H_{N/2}^{\text{rep}}$ can be expressed in terms of the following protograph

$$
H_{N/2}^{\text{rep}} = \mathfrak{B}([\lambda_{N/2}^0 + \lambda_{N/2}^1]).
\tag{30}
$$

Accepted in ⟨ ⟩uantum 2023-03-12, click title to verify. Published under CC-BY 4.0.
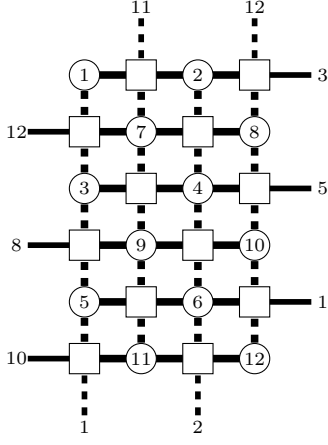
9

Figure 6: The $[[12, 2, 3]]$ XZZX twisted toric code. The boundary edges have been twisted so that each boundary check node now connects to the qubit shifted one position below on the opposite side of the lattice. This increases the length of the non-trivial logical operators, and results in an improvement in the minimum distance of the code from $d = 2$ to $d = 3$.

Here $N = 2n_1 n_2$ where n_1 and n_2 are the vertical and horizontal widths of the lattice respectively. By inspection of the factor graph in Fig. 6, we see that the X-parity checks connect qubit i to qubit $(i + n_2) \mod N/2$. Using this periodicity, we find that

$$H_1 \otimes \mathbb{1}_{n_2} = \mathfrak{B}([\lambda_{N/2}^0 + \lambda_{N/2}^{n_2}]). \quad (31)$$

The protograph representation of Eq. (29) is then given by

$$A = [A_X | A_Z] = \begin{bmatrix} 0 & A_1^T & A_2 & 0 \\ A_1 & 0 & 0 & A_2^T \end{bmatrix}, \quad (32)$$

where $A_1 = [\lambda_{N/2}^0 + \lambda_{N/2}^{n_2}]$ and $A_2 = [\lambda_{N/2}^0 + \lambda_{N/2}^1]$. We now note that $A_k = A_k \otimes E_1 = E_1 \otimes A_k$, where where E_1 is a 1×1 instance of the identity protograph defined in Eq. (18). Using this relation, we can rewrite Eq. (32) as follows

$$A = \begin{bmatrix} 0 & A_1^T \otimes E_1 & E_1 \otimes A_2 & 0 \\ A_1 \otimes E_1 & 0 & 0 & E_1 \otimes A_2^T \end{bmatrix}. \quad (33)$$

Now, note the similarity of the above to the lifted product defined in Eq. (21). The protograph for the XZZX toric code, Eq. (33), is simply the lifted product of the 1×1 seed protographs, A_1 and A_2, followed by Hadamard rotation on the second block of $N/2$ qubits. It is clear that this form of the lifted product encapsulates both the stabiliser redefinition and boundary twists that give the XZZX twisted toric code such high performance under biased noise.

4.6 The bias-tailored lifted product

In the previous section we saw that the XZZX twisted toric code can be described as a special case of the

lifted product applied to two 1×1 protographs. However, we need not be restricted to this special case: any combination of seed protographs A_1 and A_2 can be used to construct quantum LDPC codes. Given two seed protographs, $A_1 \in \mathbb{A}_L^{m_1' \times n_1'}$ with dimensions $m_1' \times n_1'$ and $A_2 \in \mathbb{A}_L^{m_2' \times n_2'}$ with dimensions $m_2' \times n_2'$, we can define a bias-tailored lifted product code with the following protograph

$$A_Q = [A_X | A_Z]$$
$$= \begin{bmatrix} 0 & (A_1)^T \otimes E_{m_2'} & E_{n_1'} \otimes A_2 & 0 \\ A_1 \otimes E_{n_2'} & 0 & 0 & E_{m_1'} \otimes (A_2)^T \end{bmatrix}. \quad (34)$$

where E_m is a $m \times m$ instance of the identity protograph defined in Eq. (18). The above defines a non-CSS code with a parity check matrix given by $\mathfrak{B}(A_Q)$.

Example 4.1. We now consider a bias-tailored lifted product where both A_1 and A_2 are the 4×4 protograph A given by Eq. (23). Only a Hadamard rotation differentiates the bias-tailored lifted product (see Eq. (34)) from the standard lifted product (see Eq. (21)). As such, the biased-tailored code with $L = 13$ has the same parameters $[[416, 18, d \leq 20]]$ as the lifted product code from Example 3.3. However, under infinite X-bias, only the A_Z component of the protograph is relevant. This is written explicitly for the $[[416, 18, d \leq 20]]$ code below

$$A_Z = \begin{bmatrix} A & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & A & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & A^T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & A^T & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & A^T & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & A^T \end{bmatrix}. \quad (35)$$

Here we see that the A_Z protograph consists of a set of decoupled copies of the the seed protograph A and its transpose. Both the code $\mathfrak{B}(A)$ and its transpose $\mathfrak{B}(A^T)$ have parameters $[52, 3, 26]$. Consequently the infinate bias X-distance is $d_X = 26$. This compares favourably to the distance of $d \leq 20$ for depolarising noise.

5 Numerical results

To assess the performance of bias-tailored quantum LDPC codes we numerically estimate the logical error rates under various bias regimes. To this end, we run Monte Carlo simulations to randomly generate errors weighted according to an instance of the biased noise channel defined in Eq. (24). These errors are then corrected using the recovery operation output by a
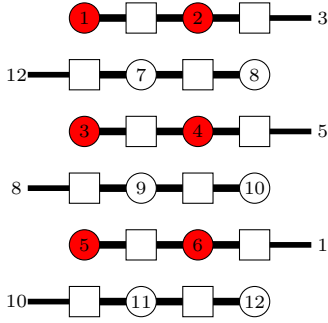
Figure 7: The H_X sub-component of the twisted $[[12, 2, 3]]$ XZZX toric code. Following the boundary twist, the minimum-weight of a non-trivial logical operator in H_Z has distance six. The qubits filled in red show an example of such a logical operator that spans across the sector one qubits.

quantum decoder. In the following, we assume we are decoding an N-qubit quantum code, with parity check matrix $H \in \mathbb{F}_2^{M \times 2N}$. The key steps in a single run of our simulation are summarised below

1. Sample a Pauli error $\mathbf{e} \in \mathbb{F}_2^{2N}$ from a distribution weighted by the chosen error channel's p_X, p_Y and p_Z probabilities.

2. Calculate the error syndrome $\mathbf{s} \in \mathbb{F}_2^M$ using Eq. (13).

3. Decode the syndrome to obtain a recovery vector $\mathbf{r} \in \mathbb{F}_2^{2N}$.

4. Calculate the residual error $\mathbf{r}' = \mathbf{e} + \mathbf{r}$.

5. Check if $\mathbf{r}'$ is a stabiliser. This amounts to verifying that $\mathbf{r}' \in \text{ROWSPACE}(H)$. If $\mathbf{r}'$ is a stabiliser then the decoding is successful. If not, then $\mathbf{r}'$ is a logical operator meaning the decoding has failed.

The fraction of failed runs gives the simulated *block error rate* P_L of the quantum code. The block error rate tells us the total probability of an error occurring on any of the logical qubits. However, it does not take into account the number of logical qubits encoded by the code. A more useful metric for quantifying code performance is the *word error rate* P_W defined as

$$P_W = 1 - (1 - P_L)^{1/K}, \tag{36}$$

where K is the number of encoded logical qubits. The word error rate can be thought of as the *logical-error-rate-per-logical-qubit* and is particularly useful for comparing quantum LDPC codes of different sizes. For small logical error rates, the word error rate approximates to $P_W \approx P_L/K$.

When decoding CSS codes, a common strategy is to treat the X- and Z-stabilisers as two separate codes. The quantum code can then be decoded using two factor graphs of size N instead of decoding the full quantum parity check matrix of size $2N$. Depending upon

the complexity scaling of the decoding algorithm, this can result in considerable reductions in runtime. Bias-tailored lifted product codes are not CSS. However, their stabilisers are locally equivalent to a CSS code, differing only by the Hadamard rotation. Using this equivalence, we have found that the decoding graph can still be decoupled into two separate components of length N. A disadvantage of this simplification is that the resultant decoder ignores correlations between the two factor graphs that arise when a Y-error occurs. To counter this, we have developed a routine that implements a Bayesian update on the error channel of the second round of decoding conditional on the output of the first round of decoding. More details about our decoding procedures can be found in Appendix D.

5.1 Decoding methods

In our simulations we use one of two decoding methods: 1) minimum-weight perfect matching (MWPM) or 2) belief propagation plus ordered statistics decoding (BP+OSD). We now briefly describe each method.

Minimum-weight perfect matching (MWPM) is a decoding method that can be applied to any code where error chains produce pair-like syndromes [44, 45]. More concisely, MWPM requires any chain-like error $\mathbf{e}$ to trigger a syndrome $\mathbf{s} = H \cdot \mathbf{e}$ of weight two such that $\text{wt}(\mathbf{s}) = 2$. On a factor graph, the triggered parity checks exist at the ends of the chain of errors. Given an ensemble of syndromes, the task of the MWPM matching decoder is to find the set of error chains (matchings) with the lowest total weight. Examples of codes with parity check matrices that are amenable to MWPM decoding include: repetition codes, surface codes, toric codes, XZZX surface codes and XZZX toric codes. Note that at the boundary surface codes produce syndromes with weight one. However, this problem can be resolved by adding *boundary nodes* to the factor graph and assigning their error probability to zero. In this work we use the `PyMatching` implementation of MWPM [46] to decode instances of the CSS and XZZX toric code.

Belief propagation plus ordered statistics decoding (BP+OSD) is a versatile decoding method for quantum LDPC codes [31]. In contrast to MWPM, BP+OSD can decode codes which produce syndromes of weight greater than two. This makes it suitable for random codes constructed from the lifted product. The first step in BP+OSD is to apply the belief propagation (BP) algorithm, a standard technique for decoding classical LDPC codes [34, 47]. Belief propagation exploits the structure of a code's factor graph to efficiently compute the probability of each bit being errored given a measured syndrome. In general, BP can decode codes in time linear to the block length provided the factor graph is sufficiently sparse and has a low-density of short-length loops. For quantum codes, the latter requirement is problematic as quan-
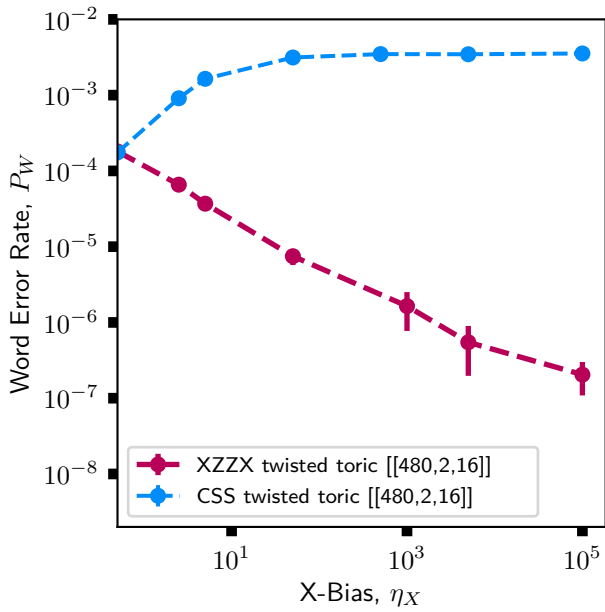
Figure 8: Decoding of the XZZX twisted toric code (red) vs. the CSS twisted toric code (blue) under increasing values of the X-bias. Both simulations were run using the MWPM decoder at a fixed physical qubit error rate of $p = 0.06$. The x-axis intersects the y-axis at $\eta_X = 0.5$ which corresponds to depolarising noise. The X-bias is increased by fixing $p_Y = p_Z$ and growing the value of p_X. Error bars show three times the standard deviation.

tum degeneracy necessarily leads to loop-like structures in the quantum factor graph. Consequently, the BP algorithm fails when degeneracy means there are multiple low-weight solutions for a certain syndrome. To counter this, the second stage of BP+OSD invokes a post-processing routine known as ordered statistics decoding (OSD) [48]. The OSD step involves inverting the parity check matrix to yield a unique solution to the decoding problem thus removing any ambiguity due to quantum degeneracy. To ensure a low-weight solution is obtained, OSD uses the output of BP to choose a high-probability subset of qubits over which the inversion is performed. Numerical simulations have shown BP+OSD to be a highly-performant decoder for hypergraph product codes [31, 49], lifted product [31] codes, 2D topological codes [49] and 3D topological codes [50]. For this work, we make use of the BP+OSD implementation included in the `LDPC` package [51].

5.2 Toric codes with twisted boundaries

We first present the finding of our numerical investigations of toric codes with twisted boundaries. As described in Section 4, such codes can be derived from the lifted product. We consider toric codes defined on a rectangular lattice with dimensions $n \times (n-1)$. These lattice parameters are chosen because they lead to the best code distances relative to the block length.

For each instance of the $n \times (n-1)$ lattice, we construct two codes: 1) the twisted CSS toric code and 2) the XZZX twisted toric code. The twisted CSS toric code has parity check matrices derived from the lifted product defined in Eq.(21) with seed protographs $A_1 = [\lambda_{N/2}^0 + \lambda_{N/2}^{(n-1)}]$ and $A_2 = [\lambda_{N/2}^0 + \lambda_{N/2}^1]$. Here, the block length is $N = 2n(n-1)$. The XZZX twisted toric code is defined from the same protographs A_1 and A_2 using the bias-tailored lifted product defined in Eq.(34). Fig. 6 depicts an example of an XZZX twisted toric on a 3×2 qubit lattice.

Fig. 8 shows the MWPM decoding curves for a CSS and XZZX twisted toric code under different bias regimes for a fixed physical error rate of $p = 0.06$. Each code is defined on a lattice with dimensions 16×15 giving code parameters $[[480, 2, 16]]$. The X-bias η_X is plotted on the x-axis and the word error rate on the y-axis. For depolarising noise, the word error rate of the two codes is equal. With increasing bias, however, the decoding performance of the two codes immediately begins to differ. For the CSS code, the word error rate begins to rise, eventually converging to a value of approximately ten times the depolarising error rate. In contrast, increasing the bias for the XZZX version of the code leads to an exponential reduction in the word error rate. This effect can be attributed to the fact that XZZX codes decouple to independent copies of their seed codes in the high-bias regime. The full quantum code distance for depolarising noise is $d = 16$, whereas the infinite bias distance for X-errors is $d_X = 240$; an improvement in code performance is therefore expected as the bias is increased. For the standard CSS version of the code, no decoupling occurs, and the distance remains $d = 16$ in all bias regimes.

5.3 Quantum LDPC codes

We numerically investigate two quantum LDPC codes constructed from the bias-tailored lifted product. The first is the $[[416, 18, d \leq 20]]$ code whose construction is outlined in Example 4.1. The second is a biased-tailored lifted product code with parameters $[[882, 24, d \leq 24]]$. This code was first presented in CSS form in [31], and the corresponding protographs can be found in Appendix B. For both codes, the performance is accessed by running simulations using the BP+OSD decoder. The lower bounds on the distances are estimated from the weights of the smallest observed logical operators during these simulations.

Fig. 9 shows the decoding plot of word error rate against X-bias for the $[[416, 18, d \leq 20]]$ lifted product code for physical error rates in the range $0.06 \leq p \leq 0.10$. As the X-bias η_X is increased we initially see an exponential decrease in the error rate. However, the decoding curve plateaus for $\eta_X > 10$. To understand why this levelling-off occurs we refer to Eq. (35) which gives the quantum protograph A_Z of
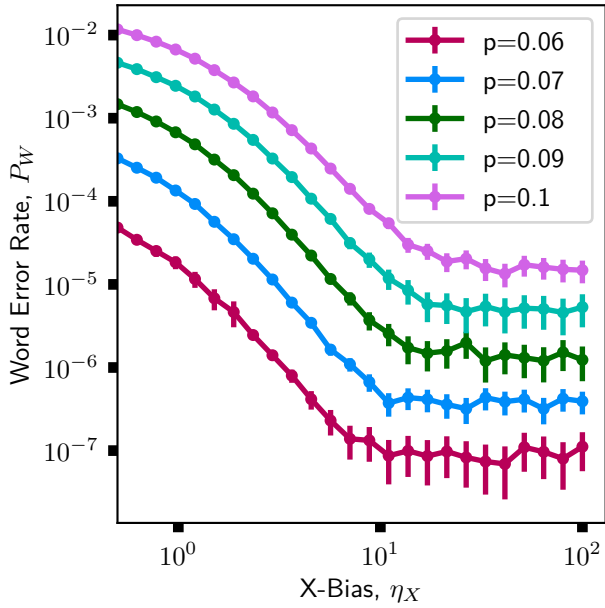
Figure 9: BP+OSD decoding of the $[[416, 18, d \leq 20]]$ bias-tailored lifted product code for physical error rates in the range $0.06 \leq p \leq 0.10$. The error bars show three times the standard deviation.
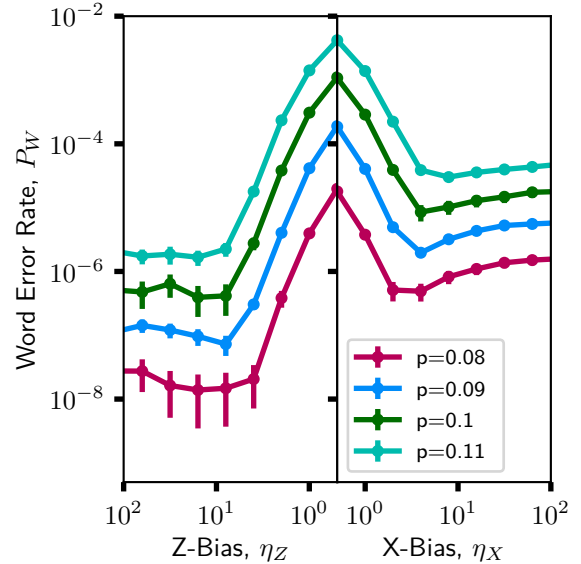

Figure 10: BP+OSD decoding of the $[[882, 24, d \leq 24]]$ bias-tailored lifted product code for values of the physical error rate in the range $0.08 \leq p \leq 0.11$. This plot is split into two parts: *Left.* The decoding plot for Z-bias, with η_Z increasing from right-to-left; *Right.* The decoding plot for X-bias, with η_X increasing from left-to-right. The two plots meet at the depolarising noise point where $\eta_X = \eta_Z = 0.5$.

the $[[416, 18, d \leq 20]]$ code in the limit of infinite X-bias. Here we see that the code has decoupled to a set of independent copies of the seed protograph A and its transpose A^T. Both A and A^T have distance 26 when converted into binary form, giving an infinite-bias X-distance of $d_X = 26$ for the $[[416, 18, d \leq 20]]$ code. As the X-bias is increased, we therefore see a reduction in the word error rate until a level commensurate with the classical performance of the seed codes is reached. From this point on, the word error rate plateaus as the code performance cannot exceed that of the original seed codes. The reason we do not see a similar plateau for the XZZX toric code in Fig. 8 is that it has an infinite-bias X-distance of $d_X = 240$. The bias at which this code will plateau is therefore well beyond the regime which can be realistically simulated.

The $[[416, 18, d \leq 20]]$ code is constructed from the bias-tailored lifted product of a pair of identical protographs. We therefore expect its performance to be the same for X- and Z-bias. In contrast, the $[[882, 24, d \leq 24]]$ code takes two different protographs as input (see Appendix B for the exact form of the seed protographs). The decoding plot for the the bias-tailored $[[882, 24, d \leq 24]]$ code is shown in Fig. 10 for increasing values of both the X- and Z-bias. For both bias-types, we again see an immediate reduction in the word error rate. However, the X-bias curves plateau at a word error rates approximately two orders of magnitude higher than the corresponding curves for Z-bias. This implies that of the two protographs, the one used to define the X-stabilisers corresponds to the better classical code.

In Fig. 10, we also note that the decoding curves of the $[[882, 24, d \leq 24]]$ reach a minimum before levelling off at an increased value. This effect arises because of the specific form of the error model we are using. As the bias is increased, the error rate for the H_Z component of the decoder increases whilst the error rate for H_X component decreases. At a certain value of η_X, the error rate for the H_X component is so low that the effective decoder failure rate is zero. As such, increasing η_X further simply results in a larger error rate for H_Z, and therefore leads to an increase in the overall word error rate.

5.4 Qubit overheads for quantum memories

The specific advantage of quantum LDPC codes over surface codes is their higher rate. Fig. 11 shows the decoding plot for the $[[416, 18, d \leq 20]]$ bias-tailored lifted product code compared to two instances of the XZZX rotated toric code with parameters $[[180, 2, 10]]$ and $[[544, 2, 17]]$ respectively. Under depolarising noise, the simulated word error rate of the $[[416, 18, d \leq 20]]$ bias-tailored lifted product code is approximately equal to that of the $[[544, 2, 17]]$ XZZX twisted toric code. However, the $[[416, 18, d \leq 20]]$ code encodes 18 logical qubits whilst the XZZX twisted toric code only encodes two. For the same word error rate, the quantum LDPC encoding therefore reduces the qubit overhead by a factor of nine compared to the toric code. Alternatively, a smaller

Accepted in 〈 Quantum 2023-03-12, click title to verify. Published under CC-BY 4.0.
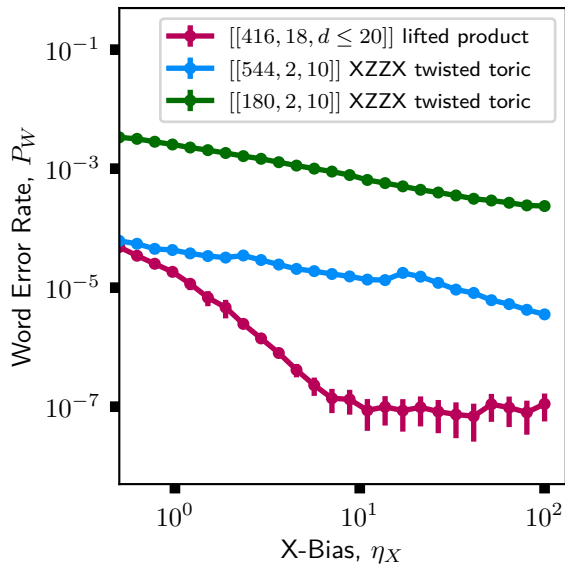
13

Figure 11: Decoding comparison between a bias-tailored lifted product code and two XZZX twisted toric codes. The toric codes are decoded using MWPM whilst the bias-tailored lifted code uses BP+OSD. For all simulations shown in this plot, the physical error rate is fixed at $p = 0.06$. Error bars show three times the standard deviation.

toric code could be used so that the 18 logical qubits can be encoded with reduced overhead. However, it is clear from decoding plot for the $[[180, 2, 10]]$ XZZX twisted toric code that this strategy imposes a severe penalty in terms of the word error rate. As the bias is increased, the word error rate of the $[[416, 18, d \leq 20]]$ code initially drops faster than the twisted toric code. Due to the plateau effect, we would expect the word rate of the lifted product code to eventually intersect with that of the $[[544, 2, 17]]$ twisted toric code. However, this would occur at an unrealistically high bias.

6 Summary & outlook

Physical qubits are typically subject to asymmetric noise where one species of error dominates over the other. As such, bias-tailoring provides a ripe opportunity for reducing the overhead of a QEC code. Prior to this work, studies conducted on the XZZX surface code have shown that it achieves remarkable performance in the high-bias regime. In this paper, we have extended these findings to show that bias-tailoring is also possible for the wider family of quantum LDPC codes.

Our principal result is the introduction of the *bias-tailored lifted product* as a method for constructing quantum codes for asymmetric noise channels. This is obtained via a modification of the stabilisers of the original lifted product defined in [28]. We first showed that the XZZX toric code can be viewed

as a special-case of the bias-tailored lifted product: both the stabiliser redefinition and boundary twists that characterise the XZZX toric code arise naturally from bias-tailored lifted product applied to two repetition codes. When applied to pairs of classical LDPC codes, the bias-tailored lifted product yields quantum LDPC codes with high-rate and good block-length-to-distance scaling. From this, we see that the bias-tailored lifted product provides a general framework for bias-tailoring, encapsulating both known examples of topological codes as well as quantum LDPC codes constructed using random methods.

We constructed examples of bias-tailored lifted product codes based on classical quasi-cyclic LDPC codes. Our Monte Carlo simulations of these codes under BP+OSD decoding showed that they outperform the toric codes for all realistic values of the bias. From this it is clear: connectivity-permitting, it is highly beneficial to consider a quantum LDPC encoding over the surface or toric code.

The performance of bias-tailored quantum LDPC codes derived from lifted products depends upon the quality of original seeds codes. This is due to fact that in the infinite-bias regime, the quantum code reduces to decoupled copies of the seed codes. As a result, it is important that the seed codes are optimised as much as possible. In this work, we have derived codes using the lifted product of quasi-cyclic codes. However, the lifted product can be applied to other families of codes. For instance, in [29], it is shown that quantum LDPC codes can be obtained using the lifted products of expander codes. It would be interesting to see whether such alternative constructions could yield codes at finite sizes (eg. < 1000 qubits) that are competitive with quasi-cyclic lifted quantum LDPC codes.

In this paper, we have considered a noise model were each qubit is subject to the same biased depolarising channel. In practice, however, each qubit on a quantum computer will be subject to a unique depolarising error channel: there is therefore the potential for the bias-tailoring to be more fine-grained. This idea has been explored for surface codes in [52] and [53]. In [52], random code deformations are investigated, whilst in [53] a scheme for individually bias-tailoring each qubit is proposed. It would be interesting to see whether similar improvements in the performance of bias-tailored quantum LDPC codes can be achieved for non-identically distributed error models. Another interesting direction for future research would be to consider the bias tailoring of higher-dimensional topological codes, as has recently been explored for various families of codes defined on 3D lattices in [54].

All the simulations in this paper were carried out under the assumption of perfect syndrome readout. A natural follow up to this work is therefore to study code performance under a more realistic *circuit-level*

noise model. As part of this, a necessary consideration is that certain quantum circuit elements can transform one error type to another. For example, a two-qubit controlled-Z gate propagates X- errors into Z-errors. Such propagation can negate the benefit of using a bias-tailored code. To counter this, it is necessary to engineer *bias-preserving* two qubit gates. It has been proposed that the XZZX surface code could retain its high-bias performance via an implementation involving Kerr-cat qubits [55]. Similarly, bias-preserving gates will need to be engineered into the design of the non-local architectures used to implement quantum LDPC codes.

In the surface code, stabiliser extraction circuits are made fault tolerant by repeating measurements and decoding over time. A similar strategy could be pursued for quantum LDPC codes. However, this is likely to introduce a large decoding overhead due to the large the number of repetitions required for fault tolerance [56]. Additionally, decoding in this context must be carried out over an extensive space-time factor graph, a complication than can impact the benefits derived from bias-tailoring [57].

An alternative strategy for constructing fault tolerant quantum error correction codes is to use a so-called *single-shot code*. Single-shot codes are designed to have redundant stabilisers that are protected by a classical code [58]. This allows imperfect measurements to be corrected, and the quantum code to be decoded using only a single round of stabiliser measurements. In [59], it was established that single-shot codes can be constructed using higher-dimensional homological products. Building on this, it has been shown that single-shot quantum LDPC codes can be efficiently decoded using BP+OSD [50, 60]. An interesting avenue for future research would be to investigate the construction of higher dimensional lifted products codes and their single-shot properties. For instance, a bias-tailored quantum LDPC code could be obtained by taking the lifted 3D hypergraph product over the ring of circulants. Such a code would be single-shot for one species of Pauli error. Extending this concept, fully single-shot codes could be obtained from lifted variants of the 4D hypergraph product.

The BP+OSD decoder serves as a general-purpose decoder for quantum LDPC codes. While the belief propagation component operates in $O(N)$ time, where N represents the block-length of the code, the ordered statistics component necessitates Gaussian elimination with an $O(N^3)$ runtime. Depending on the clock cycle of the selected qubit architecture, the OSD post-processing might present an obstacle to real-time decoding of quantum LDPC codes [61]. As a result, exploring alternative decoding strategies is crucial. One possibility is that the OSD decoding stage could be replaced by an alternative post-processing decoder such as union find [62, 63], small set-flip [64] or probabilistic flip [65]. Another strategy is to directly optimise the BP algorithm itself, for example through the use of machine learning, to derive improved update rules [66]. Finally, there are BP-variants that are specifically designed to mitigate the effects of trapping sets in quantum codes. These include generalised belief propagation [67], stabiliser inactivation [68] and memory belief propagation [69]. Such methods could reduce, or even remove, the need for resource intensive post-processing.

It has long been known, both in theory and practice, that classical codes can (near) saturate the Shannon limit. Recently, it has been proved that *optimal* families of lifted product codes exist with linear distance to block-length scaling [29]. An interesting avenue for future work would be to study the numerical performance of such codes under different bias regimes. In particular, this could cast light on the ultimate capacity of quantum channels beyond the Hashing bound.

7 Acknowledgements

7.1 People

7.2 Funding

7.3 HPC resources

7.4 Open source software

The following open source software packages were used in this project: Numpy [71]; Matplotlib [72]; Scipy [73]; LDPC [51]; BPOSD [74]; pyMatching [46]; Software for LDPC codes [75].

Accepted in 〈 〉uantum 2023-03-12, click title to verify. Published under CC-BY 4.0.

15

7.5 The planet

The carbon footprint associated with the numerical simulations in this paper are summarised below.

Numerical simulations	
Total Kernel Hours [h]	77238
Thermal Design Power Per Kernel [W]	5.75
Total Energy Consumption [kWh]	444
Average Emission Of CO_2 [kg/kWh]	0.31
Total CO_2-Emission [kg]	138

For context, 138kg is the amount of carbon that would be emitted by the average German car over a journey of approximately 1000km. The CO_2 emission per kWh used in the above table was the average value in Germany during 2020 according to European Environment Agency. The 138kg emitted will be offset by donating to Atmosfair (https://atmosfair.de), a not-for-profit that promotes, develops and finances renewable energies in over fifteen countries worldwide. Guidance on the reporting the carbon cost of scientific research can be found in [76].

References

[1] Peter W. Shor, *Scheme for reducing decoherence in quantum computer memory*, Physical Review A **52**, R2493 (1995).

[2] Joschka Roffe, *Quantum error correction: an introductory guide*, Contemporary Physics **60**, 226 (2019).

[3] P Aliferis, F Brito, D P DiVincenzo, J Preskill, M Steffen, and B M Terhal, *Fault-tolerant computing with biased-noise superconducting qubits: a case study*, New Journal of Physics **11**, 013061 (2009).

[4] Raphaël Lescanne, Marius Villiers, Théau Peronnin, Alain Sarlette, Matthieu Delbecq, Benjamin Huard, Takis Kontos, Mazyar Mirrahimi, and Zaki Leghtas, *Exponential suppression of bit-flips in a qubit encoded in an oscillator*, Nature Physics **16**, 509 (2020).

[5] Christopher Chamberland, Kyungjoo Noh, Patricio Arrangoiz-Arriola, Earl T. Campbell, Connor T. Hann, Joseph Iverson, Harald Putterman, Thomas C. Bohdanowicz, Steven T. Flammia, Andrew Keller, *et al.*, *Building a fault-tolerant quantum computer using concatenated cat codes*, (2020), arXiv:2012.04108 [quant-ph].

[6] Shruti Puri, Lucas St-Jean, Jonathan A. Gross, Alexander Grimm, Nicholas E. Frattini, Pavithran S. Iyer, Anirudh Krishna, Steven Touzard, Liang Jiang, Alexandre Blais, *et al.*, *Bias-preserving gates with stabilized cat qubits*, Science Advances **6** (2020), 10.1126/sciadv.aay5901.

[7] Juan Pablo Bonilla Ataides, David K. Tuckett, Stephen D. Bartlett, Steven T. Flammia, and Benjamin J. Brown, *The XZZX surface code*, Nature Communications **12** (2021), 10.1038/s41467-021-22274-1.

[8] Xiao-Gang Wen, *Quantum orders in an exact soluble model*, Phys. Rev. Lett. **90**, 016803 (2003).

[9] Abbas Al-Shimary, James R Wootton, and Jiannis K Pachos, *Lifetime of topological quantum memories in thermal environment*, New Journal of Physics **15**, 025027 (2013).

[10] Alexey A. Kovalev and Leonid P. Pryadko, *Improved quantum hypergraph-product LDPC codes*, in *IEEE International Symposium on Information Theory Proceedings* (2012) pp. 348–352.

[11] Héctor Bombin, Ruben S Andrist, Masayuki Ohzeki, Helmut G Katzgraber, and Miguel A Martin-Delgado, *Strong resilience of topological codes to depolarization*, Physical Review X **2**, 021004 (2012).

[12] Maika Takita, Andrew W. Cross, A.D. Córcoles, Jerry M. Chow, and Jay M. Gambetta, *Experimental demonstration of fault-tolerant state preparation with superconducting qubits*, Physical Review Letters **119** (2017), 10.1103/physrevlett.119.180501.

[13] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, *et al.*, *Quantum supremacy using a programmable superconducting processor*, Nature **574**, 505 (2019).

[14] Craig Gidney and Martin Ekerå, *How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits*, Quantum **5**, 433 (2021).

[15] Sergey Bravyi, David Poulin, and Barbara Terhal, *Tradeoffs for reliable quantum information storage in 2d systems*, Physical review letters **104**, 050503 (2010).

[16] Nouédyn Baspin and Anirudh Krishna, *Connectivity constrains quantum codes*, Quantum **6**, 711 (2022).

[17] Nicolas Delfosse, Michael E. Beverland, and Maxime A. Tremblay, *Bounds on stabilizer measurement circuits and obstructions to local implementations of quantum LDPC codes*, (2021), arXiv:2109.14599 [quant-ph].

[18] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, *Demonstration of a small programmable quantum computer with atomic qubits*, Nature **536**, 63 (2016).

[19] L. Bergeron, C. Chartrand, A. T. K. Kurkjian, K. J. Morse, H. Riemann, N. V. Abrosimov, P. Becker, H.-J. Pohl, M. L. W. Thewalt, and S. Simmons, *Silicon-integrated telecommunications photon-spin interface*, PRX Quantum **1** (2020), 10.1103/prxquantum.1.020301.

[20] P. Magnard, S. Storz, P. Kurpiers, J. Schär,

Accepted in ⟨ᑫuantum 2023-03-12, click title to verify. Published under CC-BY 4.0.

16

F. Marxer, J. Lütolf, T. Walter, J.-C. Besse, M. Gabureac, K. Reuer, *et al.*, *Microwave quantum link between superconducting circuits housed in spatially separated cryogenic systems,* Phys. Rev. Lett. **125**, 260502 (2020).

[21] Joshua Ramette, Josiah Sinclair, Zachary Vendeiro, Alyssa Rudelis, Marko Cetina, and Vladan Vuletić, *Any-to-any connected cavity-mediated architecture for quantum computing with trapped ions or rydberg arrays,* arXiv:2109.11551 [quant-ph] (2021).

[22] Nikolas P. Breuckmann and Jens Niklas Eberhardt, *Quantum low-density parity-check codes,* PRX Quantum **2** (2021), 10.1103/prxquantum.2.040101.

[23] Lawrence Z. Cohen, Isaac H. Kim, Stephen D. Bartlett, and Benjamin J. Brown, *Low-overhead fault-tolerant quantum computing using long-range connectivity,* arXiv:2110.10794 (2021), arXiv:2110.10794 [quant-ph].

[24] Shuai Shao, Peter Hailes, Tsang-Yi Wang, Jwo-Yuh Wu, Robert G Maunder, Bashir M Al-Hashimi, and Lajos Hanzo, *Survey of turbo, ldpc, and polar decoder asic implementations,* IEEE Communications Surveys & Tutorials **21**, 2309 (2019).

[25] Georgios Tzimpragos, Christoforos Kachris, Ivan B Djordjevic, Milorad Cvijetic, Dimitrios Soudris, and Ioannis Tomkos, *A survey on fec codes for 100 g and beyond optical networks,* IEEE Communications Surveys & Tutorials **18**, 209 (2014).

[26] Matthew B Hastings, Jeongwan Haah, and Ryan O'Donnell, *Fiber bundle codes: breaking the n 1/2 polylog (n) barrier for quantum LDPC codes,* in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2021) pp. 1276–1288.

[27] Nikolas P. Breuckmann and Jens N. Eberhardt, *Balanced product quantum codes,* IEEE Transactions on Information Theory **67**, 6653 (2021).

[28] Pavel Panteleev and Gleb Kalachev, *Quantum ldpc codes with almost linear minimum distance,* IEEE Transactions on Information Theory **68**, 213–229 (2022).

[29] Pavel Panteleev and Gleb Kalachev, *Asymptotically good quantum and locally testable classical ldpc codes,* in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022 (Association for Computing Machinery, New York, NY, USA, 2022) p. 375–388.

[30] Marc PC Fossorier, *Quasicyclic low-density parity-check codes from circulant permutation matrices,* IEEE Transactions on Information Theory **50**, 1788 (2004).

[31] Pavel Panteleev and Gleb Kalachev, *Degenerate quantum ldpc codes with good finite length performance,* Quantum **5**, 585 (2021).

[32] Joschka Roffe, Stefan Zohren, Dominic Horsman, and Nicholas Chancellor, *Quantum codes from classical graphical models,* IEEE Transactions on Information Theory **66**, 130 (2020).

[33] Joschka Roffe, *Simulating bias-tailored QLDPC codes,* https://github.com/quantumgizmos/bias_tailored_qldpc.

[34] Frank R Kschischang, Brendan J Frey, Hans-Andrea Loeliger, *et al.*, *Factor graphs and the sum-product algorithm,* IEEE Transactions on Information Theory **47**, 498 (2001).

[35] Lindsay N Childs, *A concrete introduction to higher algebra* (Springer, 2009).

[36] A. R. Calderbank and Peter W. Shor, *Good quantum error-correcting codes exist,* Phys. Rev. A **54**, 1098 (1996).

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

[38] A. M. Steane, *Active stabilization, quantum computation, and quantum state synthesis,* Physical Review Letters **78**, 2252 (1997).

[39] Jean-Pierre Tillich and Gilles Zémor, *Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the block-length,* IEEE Transactions on Information Theory **60**, 1193 (2013).

[40] Armanda O. Quintavalle and Earl T. Campbell, *Reshape: A decoder for hypergraph product codes,* IEEE Transactions on Information Theory **68**, 6569 (2022).

[41] Xiao-Yu Hu, E. Eleftheriou, and D.-M. Arnold, *Progressive edge-growth tanner graphs,* in *IEEE Global Telecommunications Conference*, Vol. 2 (2001) pp. 995–1001 vol.2.

[42] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill, *Topological quantum memory,* Journal of Mathematical Physics **43**, 4452 (2002).

[43] Ben Criger and Imran Ashraf, *Multi-path Summation for Decoding 2D Topological Codes,* Quantum **2**, 102 (2018).

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

[45] Vladimir Kolmogorov, *Blossom v: a new implementation of a minimum cost perfect matching algorithm,* Mathematical Programming Computation **1**, 43 (2009).

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

[47] David JC MacKay and Radford M Neal, *Near shannon limit performance of low density parity check codes,* Electronics Letters **33**, 457 (1997).

[48] Marc PC Fossorier, *Iterative reliability-based decoding of low-density parity check codes,* IEEE

Journal on Selected Areas in Communications **19**, 908 (2001).

[49] Joschka Roffe, David R. White, Simon Burton, and Earl Campbell, *Decoding across the quantum low-density parity-check code landscape,* Phys. Rev. Research **2**, 043423 (2020).

[50] Armanda O. Quintavalle, Michael Vasmer, Joschka Roffe, and Earl T. Campbell, *Single-shot error correction of three-dimensional homological product codes,* PRX Quantum **2** (2021), 10.1103/prxquantum.2.020340.

[51] Joschka Roffe, *LDPC: Python tools for low density parity check codes,* https://pypi.org/project/ldpc/ (2022).

[52] Arpit Dua, Aleksander Kubica, Liang Jiang, Steven T. Flammia, and Michael J. Gullans, *Clifford-deformed surface codes,* (2022), 10.48550/ARXIV.2201.07802.

[53] Konstantin Tiurev, Peter-Jan H. S. Derks, Joschka Roffe, Jens Eisert, and Jan-Michael Reiner, *Correcting non-independent and non-identically distributed errors with surface codes,* (2022), 10.48550/ARXIV.2208.02191.

[54] Eric Huang, Arthur Pesah, Christopher T. Chubb, Michael Vasmer, and Arpit Dua, *Tailoring three-dimensional topological codes for biased noise,* (2022).

[55] Andrew S. Darmawan, Benjamin J. Brown, Arne L. Grimsmo, David K. Tuckett, and Shruti Puri, *Practical quantum error correction with the XZZX code and kerr-cat qubits,* PRX Quantum **2** (2021), 10.1103/prxquantum.2.030345.

[56] Theerapat Tansuwannont, Balint Pato, and Kenneth R. Brown, *Adaptive syndrome measurements for shor-style error correction,* (2023), arXiv:2208.05601 [quant-ph].

[57] Oscar Higgott, Thomas C. Bohdanowicz, Aleksander Kubica, Steven T. Flammia, and Earl T. Campbell, *Fragile boundaries of tailored surface codes and improved decoding of circuit-level noise,* (2022), arXiv:2203.04948 [quant-ph].

[58] Héctor Bombín, *Single-shot fault-tolerant quantum error correction,* Physical Review X **5**, 031043 (2015).

[59] Earl Campbell, *A theory of single-shot error correction for adversarial noise,* Quantum Science and Technology (2019), 10.1088/2058-9565/aafc8f.

[60] Oscar Higgott and Nikolas P. Breuckmann, *Improved single-shot decoding of higher dimensional hypergraph product codes,* (2022), arXiv:2206.03122 [quant-ph].

[61] Javier Valls, Francisco Garcia-Herrero, Nithin Raveendran, and Bane Vasić, *Syndrome-based min-sum vs osd-0 decoders: Fpga implementation and analysis for quantum ldpc codes,* IEEE Access **9**, 138734 (2021).

[62] Nicolas Delfosse, Vivien Londe, and Michael E. Beverland, *Toward a union-find decoder for quantum ldpc codes,* IEEE Transactions on Information Theory **68**, 3187 (2022).

[63] Lucas Berent, Lukas Burgholzer, and Robert Wille, *Software tools for decoding quantum low-density parity-check codes,* in *Proceedings of the 28th Asia and South Pacific Design Automation Conference,* ASPDAC '23 (Association for Computing Machinery, New York, NY, USA, 2023) p. 709–714.

[64] Antoine Grospellier, Lucien Grouès, Anirudh Krishna, and Anthony Leverrier, *Combining hard and soft decoders for hypergraph product codes,* (2020), arXiv:2004.11199.

[65] T. R. Scruby and K. Nemoto, *Local probabilistic decoding of a quantum code,* arXiv:2212.06985 [quant-ph] (2023).

[66] Ye-Hua Liu and David Poulin, *Neural belief-propagation decoders for quantum error-correcting codes,* Physical Review Letters **122** (2019), 10.1103/physrevlett.122.200501.

[67] Josias Old and Manuel Rispler, *Generalized belief propagation algorithms for decoding of surface codes,* arXiv:2212.03214 [quant-ph] (2022).

[68] Julien Du Crest, Mehdi Mhalla, and Valentin Savin, *Stabilizer inactivation for message-passing decoding of quantum ldpc codes,* in *2022 IEEE Information Theory Workshop (ITW)* (2022) pp. 488–493.

[69] Kao-Yueh Kuo and Ching-Yi Lai, *Exploiting degeneracy in belief propagation decoding of quantum codes,* npj Quantum Information **8** (2022), 10.1038/s41534-022-00623-2.

[70] Loris Bennett, Bernd Melchers, and Boris Proppe, *Curta: A general-purpose high-performance computer at ZEDAT, freie universität berlin,* (2020), 10.17169/REFUBIUM-26754.

[71] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux, *The numpy array: a structure for efficient numerical computation,* Computing in Science & Engineering **13**, 22 (2011).

[72] J. D. Hunter, *Matplotlib: A 2d graphics environment,* Computing in Science & Engineering **9**, 90 (2007).

[73] Virtanen et al. and SciPy 1. 0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,* Nature Methods **17**, 261 (2020).

[74] Joschka Roffe, *BP+OSD: Belief propagation with ordered statistics post-processing for decoding quantum LDPC codes,* (2020), https://github.com/quantumgizmos/bp_osd.

[75] Radford M. Neal, *Software for low density parity check codes,* (2012), http://radfordneal.github.io/{LDPC}-codes/.

[76] *Scientific CO2nduct, raising awareness for the climate impact of science*, https://scientific-conduct.github.io.

[77] Claude Elwood Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27**, 379 (1948).

[78] Robert Gallager, *Low-density parity-check codes*, IRE Transactions on Information Theory **8**, 21 (1962).

[79] Claude Berrou and Alain Glavieux, *Near optimum error correcting coding and decoding: Turbo-codes*, IEEE Transactions on Communications **44**, 1261 (1996).

[80] Erdal Arikan, *Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels*, IEEE Transactions on Information Theory **55**, 3051 (2009).

[81] Charles H. Bennett, David P. DiVincenzo, John A. Smolin, and William K. Wootters, *Mixed-state entanglement and quantum error correction*, Phys. Rev. A **54**, 3824 (1996).

[82] David P. DiVincenzo, Peter W. Shor, and John A. Smolin, *Quantum-channel capacity of very noisy channels*, Phys. Rev. A **57**, 830 (1998).

[83] Peter W. Shor and John A. Smolin, *Quantum error-correcting codes need not completely reveal the error syndrome*, (1996), arXiv:quant-ph/9604006 [quant-ph].

A   Parity check matrices

In Example 3.2, a hypergraph product code is constructed from seed codes with the parameters $[16, 4, 6]$. The parity check matrix H for these seed codes is given below:

$$
H = \begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}. \tag{37}
$$

B   Protographs

The $[[882, 24, d \le 24]]$ code is constructed from the bias-tailored lifted product with $L = 63$ of the following seed protographs

$$
A_1 = \begin{bmatrix} \lambda_L^0 + \lambda_L^1 + \lambda_L^6 \end{bmatrix} \tag{38}
$$

$$
A_2 = \begin{bmatrix}
\lambda_L^{36} & 0 & 0 & 0 & 0 & \lambda_L^0 & \lambda_L^9 \\
\lambda_L^9 & \lambda_L^{36} & 0 & 0 & 0 & 0 & \lambda_L^0 \\
\lambda_L^0 & \lambda_L^9 & \lambda_L^{36} & 0 & 0 & 0 & 0 \\
0 & \lambda_L^0 & \lambda_L^9 & \lambda_L^{36} & 0 & 0 & 0 \\
0 & 0 & \lambda_L^0 & \lambda_L^9 & \lambda_L^{36} & 0 & 0 \\
0 & 0 & 0 & \lambda_L^0 & \lambda_L^9 & \lambda_L^{36} & 0 \\
0 & 0 & 0 & 0 & \lambda_L^0 & \lambda_L^9 & \lambda_L^{36}
\end{bmatrix}. \tag{39}
$$

The upper-bound on the minimum distance is an estimate based on the lowest-weight logical operator observed during simulation with the BP+OSD decoder. This code is equivalent to the CSS *generalised hypergraph product* code of the same size first presented in [31].

C   The Hashing bound under biased-noise

Given a noisy channel, what is the maximum amount of information that can be reliably transferred using error correction? In the classical setting, the answer to this question is given by the Shannon noisy coding theorem [77]. For a given error channel, the Shannon noisy coding theorem tells us that *good* error correction codes exist with the ability to arbitrarily suppress the logical error rate. However, the rate $r = k/n$ of such good codes must fall below a certain value known as the Shannon limit. To illustrate, we consider the simple case of the classical binary symmetric error channel where each bit is independently subject to a bit-flip with probability p_X. For this channel, the Shannon limit r_S is given by

$$
r_S = 1 - \mathcal{H}(p_X), \tag{40}
$$

where $\mathcal{H}(p_X)$ is the binary entropy defined as

$$
\mathcal{H}(p_X) = -(1-p_X)\log_2(1-p_X) - p_X \log_2(p_X). \tag{41}
$$

The Shannon limit implies that *perfect* error correction is in principle possible with error correction codes with code rate $r \le r_S$, whilst it is impossible with code rate $r > r_S$. Various classical error correction codes – including LDPC, turbo, and polar codes – are known to be *optimal* in the sense that they operate at rates approaching the Shannon limit whilst being efficiently decodable [47, 78–80].

The Shannon noisy coding theorem can be directly extended to quantum channels. This provides a Shannon limit for stabiliser codes referred to as the Hashing bound. For the noisy quantum channel $\mathcal{E}_Q(\rho)$ defined in Eq. (24) the Hashing bound r_H is given by [81]

$$
r_H = 1 - \mathcal{H}_Q(p). \tag{42}
$$

Here, $\mathcal{H}_Q(p)$ is the Pauli entropy function defined as

$$
\begin{aligned}
\mathcal{H}_Q(p) = &-(1-p)\log_2(1-p) \\
&-p_X\log_2(p_X) - p_Y\log_2(p_Y) - p_Z\log_2(p_Z),
\end{aligned}
\tag{43}
$$

where p is the total physical error rate $p = p_X + p_Y + p_Z$. It should be noted that the Hashing bound is derived using methods from classical information theory: quantum effects such as entanglement and degeneracy, which increase the actual capacity of the quantum channel [82], are not accounted for. As such, the Hashing bound should be treated as a lower-limit on the achievable rate of reliable information transfer over a quantum channel. In fact, in certain settings, stabiliser codes have been shown to achieve logical error rate suppression beyond the Hashing bound [7, 83]. Quantifying the true capacity of a quantum channel remains an open problem.

Given an existing error correction code with a known rate it is possible to calculate its operational regime by calculating the Hashing probability $p_H(r)$. The Hashing probability is the value obtained for the total physical probability p when r_H in Eq. (42) is set to a fixed value $r_H = r$, which is readily obtained using a numerical solver. The Hashing probability $p_H(r)$ tells us the physical error rate below which it is possible to arbitrarily suppress the logical error rate.

The toric code and its topological cousins are often benchmarked in terms of the code threshold p_{TH}. Provided the physical error rate of the qubits is below the threshold $p < p_{TH}$, increasing the size of a code will reduce the logical error rate p_L. In the sub-threshold regime, and in the limit of large block lengths where the code rate $r \to 0$, we expect the logical error rate to be arbitrarily suppressed $\lim_{r\to 0} p_L = 0$. For these codes, an estimate of the code threshold can therefore be derived from the *zero-rate* Hashing probability $p_H(0)$.

We now explore how adjusting the qubit bias affects the Hashing limit of the quantum channel. Fig. 12 shows the Hashing probability plotted against the X-bias for code error rates between $R = 10\%$ and $R = 0$. Here the X-bias is increased setting $p_Y = p_Z$ and raising the value of p_X whilst keeping the total physical error rate fixed $p = p_X + p_Y + p_Z$. The red line shows the zero-rate Hashing probability and therefore gives an estimate of the code threshold. At the origin, which corresponds to depolarising noise where $p_X = p_Y = p_Z$, the zero-rate Hashing limit is 18.9%. As the bias grows, the Hashing limit rises until converging to a value of 50%. In the infinite-bias regime, it is clear that the Hashing bound defined in Eq. (42) is equivalent to the classical Shannon limit in Eq. (40).

Analysing the quantum noise model in terms of the Hashing bound implies that there is much to be gained from accounting for noise asymmetry in the design of quantum codes. Up to this point, we have explored the zero-rate Hashing bound that only applies to codes, such as the toric code, which have vanishing
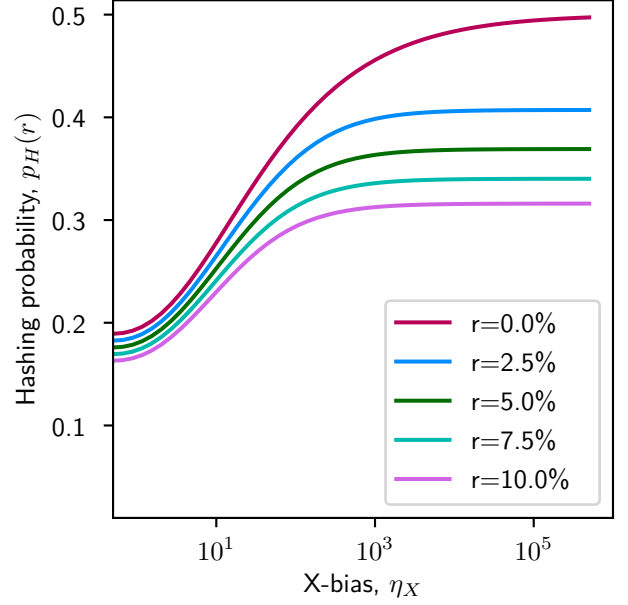

Figure 12: Plots of of the hashing probability against the X-bias for code rates between $R = 10\%$ and $R = 0\%$. The x-axis intersects the y-axis at $\eta_X = 0.5$ which corresponds to depolarising noise. The Hashing probabilities were obtained by numerically solving Eq. (42) for p for fixed values of the rate $r_H = r$.

rate in the asymptomatic limit. In addition to the zero-rate line, Fig. 12 also plots the Hashing probability for various finite values of the rate. For all rates, we see an improvement in the Hashing capacity as the bias is increased. As such, we also expect bias-tailoring to be useful for codes, such as quantum LDPC codes, that have finite rate in the asymptotic limit.

D Simulation algorithm

In this Appendix we describe implementation details of our Monte Carlo simulations of quantum LDPC decoding. We present all algorithms in pseudo-code. The actual simulation code can be found in the external repository for this project [33].

D.1 Two-stage decoding of Hadamard rotated codes

For CSS codes, it is possible to implement a two-stage decoding procedure where the X- and Z-stabilisers are treated as separate codes. The stabilisers of the bias-tailored codes explored this work differ from their *standard* CSS code counterparts by a Hadamard rotation on the sector-two qubits. The resultant codes are non-CSS which would ordinarily necessitate having to decode the entire quantum parity check matrix in a single stage. Fortunately, for the specific case of the Hadamard rotated codes, a standard CSS two-stage

decoding routine remains possible. To illustrate, we first consider the effect the Hadmard rotation has on the syndrome equations. We take the case of a standard CSS code obtained from a product construction (eg. hypegraph product, lifted product etc.) such that its qubits can be separated into sector-one and sector-two qubits. In this setting, the quantum parity check matrix has the form

$$H_{\mathrm{CSS}} = \begin{bmatrix} 0 & H_Z' \\ H_X' & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & H_{Z1} & H_{Z2} \\ H_{X1} & H_{X2} & 0 & 0 \end{bmatrix}.$$

Following a Hadmard rotation, the above parity check matrix is transformed to

$$H_{\mathrm{HR}} = \begin{bmatrix} 0 & H_{Z2} & H_{Z1} & 0 \\ H_{X1} & 0 & 0 & H_{X2} \end{bmatrix}.$$

To compute the syndrome we apply the Hadamard rotated stabilisers H_{HR} to the error vector $\mathbf{e} = (\mathbf{e_X}|\mathbf{e_Z})$, where the X- and Z-components are split into sector-one and sector-two qubits such that $\mathbf{e}_X = (\mathbf{e}_{X1}, \mathbf{e}_{X2})$ and $\mathbf{e}_Z = (\mathbf{e}_{Z1}, \mathbf{e}_{Z2})$. The syndrome equation then reads

$$(\mathbf{s}_X|\mathbf{s}_Z) = H_{\mathrm{HR}} \star \mathbf{e} =$$
$$(H_{Z1} \cdot \mathbf{e}_{X1} + H_{Z2} \cdot \mathbf{e}_{Z2}|H_{X1} \cdot \mathbf{e}_{Z1} + H_{X2} \cdot \mathbf{e}_{X2}).$$

It is now clear that decoding for the Hadamard rotated codes decouples into two problems

$$\begin{aligned} \mathbf{s}_X &= H_Z' \cdot (\mathbf{e}_{X1}, \mathbf{e}_{Z2}), \\ \mathbf{s}_Z &= H_X' \cdot (\mathbf{e}_{Z1}, \mathbf{e}_{X2}), \end{aligned} \qquad (44)$$

where H_X' and H_Z' are the CSS parity check matrices prior to the Hadamard rotation. From this, we see that for the purposes of simulation it is sufficient to simply Hadamard rotate the error vector whilst retaining the stabilisers in their original CSS form. In practice, this is achieved by modifying the error model that is input to the simulation algorithm.

D.2 Main decoding simulation

Algorithm 1 gives a pseudocode description of the Monte Carlo simulation used to access the performance of the quantum LDPC codes in this paper. Lines 9-12 show how the Hadamard rotation is applied by modifying the original error model so that the $N - N_1$ qubits in sector-two have p_X swapped with p_Z. This ensures that the randomly generated error vectors have the Hadamard rotated form from Eq (44). In our simulations one of two decoding methods are used: 1) minimum-weight perfect matching (MWPM); 2) belief propagation plus ordered statistics decoding (BP+OSD). These decoders are described qualitatively in Section 5.1. Both decoding methods act over the factor graph of the code and accept prior information about the error model.

Algorithm 1 SIMULATION: DECODING BIAS-TAILORED CODES

Input: Decoder; CSS parity check matrices H_X' and H_Z'; CSS logical operator matrices L_X' and L_Z'; Code block length N; Number of qubits in sector-one N_1; Qubit error rates for Pauli errors $\{p_X, p_Y, p_Z\}$.

Output: Decoding Success or Failure

1: $\mathbf{e}_X \leftarrow \mathbf{0}$ ▷ Qubit X-error
2: $\mathbf{e}_Z \leftarrow \mathbf{0}$ ▷ Qubit Z-error
3: $\mathbf{p}_X \leftarrow \mathbf{0}$ ▷ Priors for $\mathbf{e}_X$
4: $\mathbf{p}_Z \leftarrow \mathbf{0}$ ▷ Priors for $\mathbf{e}_Z$

       ▷ Hadamard rotate error model
5: **for** $j \leftarrow 1$ **to** N_1 **do**
6:      $\mathbf{p}_X[j] \leftarrow p_X + p_Y$ ▷ X-priors on sector-1
7:      $\mathbf{p}_Z[j] \leftarrow p_Z + p_Y$ ▷ Z-prior on sector-1
8: **end for**
9: **for** $j \leftarrow N_1 + 1$ **to** N **do**
10:     $\mathbf{p}_X[j] \leftarrow p_Z + p_Y$ ▷ X-priors on sector-2
11:     $\mathbf{p}_Z[j] \leftarrow p_X + p_Y$ ▷ Z-prior on sector-2
12: **end for**

       ▷ Generate random qubit error
13: **for** $j \leftarrow 1$ **to** N **do**
14:     $\zeta \leftarrow$ random number between 0 and 1
15:     **if** $\zeta < \mathbf{p}_X[j]$ **then**
16:        $\mathbf{e}_X[j] \leftarrow 1$
17:        $\mathbf{e}_X[j] \leftarrow 0$
18:     **else if** $\mathbf{p}_X[j] < \zeta < \mathbf{p}_X[j] + \mathbf{p}_Z[j]$ **then**
19:        $\mathbf{e}_X[j] \leftarrow 0$
20:        $\mathbf{e}_Z[j] \leftarrow 1$
21:     **else if** $\mathbf{p}_X[j] + \mathbf{p}_Z[j] < \zeta < \mathbf{p}_X[j] + \mathbf{p}_Z[j] + \mathbf{p}_Y[j]$ **then**
22:        $\mathbf{e}_X[j] \leftarrow 1$
23:        $\mathbf{e}_Z[j] \leftarrow 1$
24:     **end if**
25: **end for**

       ▷ Calculate syndromes
26: $\mathbf{s}_X = H_Z' \cdot \mathbf{e}_X$ ▷ X-syndrome
27: $\mathbf{s}_Z = H_X' \cdot \mathbf{e}_Z$ ▷ Z-syndrome

       ▷ X-error decoding
28: Use the decoder with priors $\mathbf{p}_X$ to obtain recovery $\mathbf{r}_X$ s.t. $H_Z' \cdot \mathbf{r}_X = \mathbf{s}_X$

       ▷ Channel update subroutine. The priors $\mathbf{p}_Z$ for the second round of decoding can be updated based on the output of the first round of decoding. This helps account for correlations between the X- and Z-decoding rounds that arise due to independent Y-errors. See Algorithm 2 for pseudocode of this subroutine.

       ▷ Z-error decoding
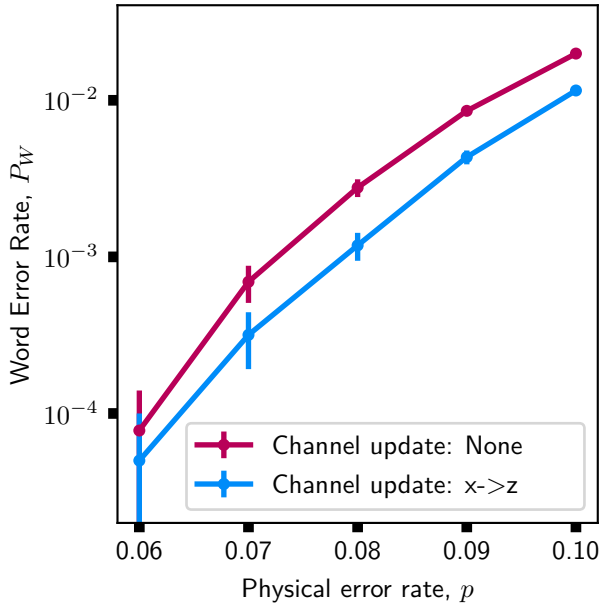29: Use the decoder with priors $\mathbf{p}_Z$ to obtain recovery

Figure 13: BP+OSD decoding of the $[[416, 18, d \leq 20]]$ quantum LDPC code with and without the channel update sub-routine described in Algorithm 2.

$$\mathbf{r}_Z \text{ s.t. } H'_X \cdot \mathbf{r}_Z = \mathbf{s}_Z$$

▷ Check for logical errors
30: $\mathbf{r}'_X = \mathbf{r}_X + \mathbf{e}_X$ ▷ Residual X-error
31: $\mathbf{r}'_Z = \mathbf{r}_Z + \mathbf{e}_Z$ ▷ Residual Z-error
32: **if** $L'_X \cdot \mathbf{r}'_Z \neq \mathbf{0}$ **then**
33:     **return** Failure
34: **else if** $L'_Z \cdot \mathbf{r}'_X \neq \mathbf{0}$ **then**
35:     **return** Failure
36: **else**
37:     **return** Success
38: **end if**

D.3 Accounting for correlated errors

Pauli-Y errors introduce correlations between the $\mathbf{e}_X$ and $\mathbf{e}_Z$ error vectors. Failing to account for these correlations can reduce the logical error rate of the decoding. Consider the scenario, as in Algorithm 1, where the X-errors are decoded before the Z-errors. Prior to the first round of decoding, we assign the probability that bit-i is set to one in the error vector $\mathbf{e}_X$ to be $\mathbf{p}_X[i] = p_X + p_Y$: i.e. this is the combined probability that either an X- or a Y-error occurs. Now, imagine that the decoder tells us that the bit-i in $\mathbf{e}_X$ is zero $\mathbf{e}_X[i] = 0$. In the second round of decoding, the probability that bit-i is $\mathbf{e}_Z[i] = 1$ is now given by

$$\mathbf{p}_Z[i] = \frac{p_Z}{1 - p_X - p_Y}. \tag{45}$$

In the above, probability of $\mathbf{p}_Z[i]$ is adjusted to account for the fact that that Y-errors are no longer possible. Similarly, if the outcome of the fist round of decoding is $\mathbf{e}_X[i] = 1$ then the the corresponding

probability $\mathbf{p}_Z[i]$ is updated to

$$\mathbf{p}_Z[i] = \frac{p_Y}{p_X + p_Y}. \tag{46}$$

These channel updates are implemented via the subroutine described in pseudocode below

---

Algorithm 2 CHANNEL UPDATE SUBROUTINE

Input: Stage 1 decoding output $\mathbf{r}_X$; Code block length N; Qubit error rates for Pauli errors $\{p_X, p_Y, p_Z\}$.
Output: Updated Z-priors $\mathbf{p}_Z$
1: $\mathbf{p}_Z \leftarrow \mathbf{0}$ ▷ Priors for $\mathbf{e}_Z$
2: **for** $j \leftarrow 1$ **to** N **do**
3:     **if** $\mathbf{r}_X[i] = 0$ **then**
4:         $\mathbf{p}_Z[i] = p_Z/(1 - p_X - p_Y)$
5:     **else if** $\mathbf{r}_X[i] = 1$ **then**
6:         $\mathbf{p}_Z[i] = p_Y/(p_X + p_Y)$
7:     **end if**
8: **end for**
9: **return** $\mathbf{p}_Z$

In this section, we have described channel updates from X->Z. However, updates in the reverse direction can easily be derived by swapping the roles of p_X with p_Z in Eqs. (45-46). For the simulations in this work, we implemented channel updates for the codes decoded using BP+OSD. This was possible, as the channel information for BP+OSD can be updated on-the-fly without affecting the runtime. We did not implement channel updates with the MWPM decoder, as updating the channel information for this decoder necessitates re-running a pre-processing routine (Dijkstra's algorithm) that compiles a matching database.

Fig. 13 shows the word error rate against the physical depolarising error rate for the $[[416, 18, d \leq 20]]$ quantum LDPC code decoded using BP+OSD. The red line shows the case when no channel update is performed, whilst for the blue line a channel update from X->Z performed. From this it is clear that the channel update subroutine improves the word error rate for all values of the physical error rate.