Decoding across the quantum low-density parity-check code landscape

Joschka Roffe,[1,*] David R. White,[1] Simon Burton,[2] and Earl Campbell[1]
[1]*Department of Physics & Astronomy, University of Sheffield, Sheffield S10 2TN, United Kingdom*
[2]*Department of Physics & Astronomy, University College London, London WC1E 6BT, United Kingdom*

(Received 19 August 2020; accepted 23 November 2020; published 28 December 2020)

We show that belief propagation combined with ordered statistics post-processing is a general decoder for quantum low-density parity-check codes constructed from the hypergraph product. To this end, we run numerical simulations of the decoder applied to three families of hypergraph product code: topological codes, fixed-rate random codes, and a new class of codes that we call semitopological codes. Our new code families share properties of both topological and random hypergraph product codes, with a construction that allows for a finely controlled trade-off between code threshold and stabilizer locality. Our results indicate thresholds across all three families of hypergraph product code, and provide evidence of exponential suppression in the low error regime. For the toric code, we observe a threshold in the range $9.9\% \pm 0.2\%$. This result improves upon previous quantum decoders based on belief propagation, and approaches the performance of the minimum-weight perfect-matching algorithm. We expect semitopological codes to have the same threshold as toric codes, as they are identical in the bulk, and we present numerical evidence supporting this observation.

DOI: 10.1103/PhysRevResearch.2.043423

I. INTRODUCTION

Any scalable computer architecture must be robust against hardware imperfections. In quantum computing, where qubits are realized as fragile quantum two-level systems, fault tolerance necessitates active error correction [1–5]. A quantum error correction *code* specifies an encoding in which quantum data are distributed across a larger space of qubits to create a logical qubit state. Errors are detected on the logical state via a series of nondestructive stabilizer measurements (quantum parity checks) yielding an error *syndrome*. This syndrome information is processed by a *decoding* algorithm to determine the best *recovery* operation to return the encoded quantum information to its uncorrupted state. All three stages of the error correction cycle—syndrome measurement, decoding, and recovery—must be performed within a short time frame before the qubits irreversibly decohere. Performing the decoding in real time is a computationally intensive inference problem, with realistic resource estimates showing a need for terabytes of syndrome information to be processed per second [6]. As such, efficient decoding algorithms are necessary to allow quantum error correction to be performed while maintaining realistic demands on classical co-processors [7].

Low-density parity-check (LDPC) codes are a ubiquitous classical error correction protocol [8], finding use, for example, in the recent 5G communication standard [9]. The specific advantage of LDPC codes is that they can be decoded using an algorithm from probabilistic graph theory known as iterative belief propagation (BP) [10]. The BP algorithm exploits the structure of the error correction code to solve the decoding inference problem in time linear in the code block length [11]. For certain LDPC codes, BP decoding enables error correction at close to the Shannon capacity, the theoretical upper bound on the rate of information transfer along a noisy channel [12,13].

Quantum LDPC (QLDPC) codes can be constructed from classical LDPC codes using the hypergraph product framework due to Tillich and Zemor [14]. The hypergraph product translates the parity check sequences of a classical *parent* code into a set of commuting stabilizers that define a quantum code. The most commonly studied hypergraph product codes fall into one of two types: *topological* QLDPC codes and *random* or *expander* QLDPC codes.

In contrast to classical LDPC codes, there is no established decoder that works generally for all QLDPC codes. For purely 2D topological codes, the minimum-weight perfect-matching algorithm achieves a threshold [15] that is close to the theoretically maximum possible value derived from statistical mechanics arguments [16]. For random QLDPC codes with the expansion property [17–19], the small set-flip (SSF) decoder has a theoretically proven threshold [18] that has been verified numerically [20]. Furthermore, in a recent study by Grospellier *et al.* [21], it was shown that the performance of the SSF decoder can be improved by combining it with the classical BP algorithm. This two-stage BP + SSF decoder exhibits a higher code threshold, in addition to being applicable to a wider range of random QLDPC codes than the SSF decoder alone.

In this paper, we consider another two-stage quantum decoder, first proposed by Panteleev and Kalachev [22], that combines BP with a post-processing technique known as ordered statistics decoding (OSD) [23,24]. Panteleev and

*joschka@roffe.eu

Kalachev demonstrated that for many random QLDPC codes, the BP + OSD method improves decoding performance by several orders magnitude over the BP algorithm alone. In this work, we expand on the results of Panteleev and Kalachev to provide further evidence that the BP + OSD decoder is a general decoder for all QLDPC codes that can be constructed from the hypergraph product. To this end, we first propose a new class of semitopological codes which share properties of both topological and random QLDPC codes. We use this new class of codes to define a spectrum of QLDPC codes, and run numerical simulations to show that the BP + OSD decoder applies generally across it.

Topological QLDPC codes have stabilizers that can be locally embedded in some D-dimensional space [25]. The simplest example is the surface code, obtained by taking the hypergraph product of the classical repetition code. The stabilizers of the surface code are *local*, meaning they can be implemented via nearest-neighbor interactions on a 2D array of qubits [25,26]. With regard to experimental implementation, this is highly beneficial, as many qubit technologies are limited in terms of connectivity between qubits [27–29]. Another practical advantage of the surface code is that it has a high threshold [15,16,26]. The disadvantage of the surface code, and topological codes in general, is that they have poor encoding rate. The surface code, for example, encodes only a single qubit per logical block meaning its encoding rate tends to zero as the code distance is increased.

Random QLDPC codes are constructed by taking the hypergraph product of high-performance classical LDPC codes. The strength of QLDPC codes over topological codes is that they can have considerably higher encoding rates that do not tend to zero with increasing block length [17–19]. The trade-off is that random QLDPC codes have *nonlocal* stabilizer checks, typically requiring interactions between arbitrary qubit pairs. Quantum computers based on ion traps [30–33], photonic qubits [34,35], or nitrogen vacancy centers [36] promise connectivity beyond nearest neighbors. However, such prototype devices do not yet meet the connectivity requirements of high-rate random QLDPC codes. Another disadvantage of random QLDPC codes is that they appear to have lower thresholds than their topological counterparts [20–22,37,38].

The new class of semitopological codes we propose in this work allow for interpolation between local topological codes and nonlocal random QLDPC codes. The construction of semitopological codes begins by modifying a classical parent code via a process called *edge augmentation*. This involves replacing each parity check edge with a length-g section of repetition code referred to as a *chain segment*. The semitopological code is then obtained from the augmented parent code via the hypergraph product, which maps each of the chain segments to a surface-code-like patch. A semitopological code can therefore be thought of as a set of surface code patches connected to one another at their boundaries via a small number of long-range interactions. The locality of a semitopological code can be finely controlled by varying the degree to which the parent code is augmented. The ability to control connectivity makes semitopological codes promising candidates for networked surface code architectures [39].

In its unmodified form, the BP algorithm is ineffective for decoding QLDPC codes due to degenerate quantum errors. Quantum degeneracy is a uniquely quantum effect, and arises in situations where quantum superposition permits multiple equivalent solutions to the decoding problem. Panteleev and Kalachev [22] have shown that for random QLDPC codes, the problem of quantum degeneracy can be resolved by decoding using BP in conjunction with OSD post-processing. The OSD method is called when BP fails, and uses matrix inversion to resolve ambiguities in the decoding due to quantum degeneracy.

In this work, we show that in addition to random QLDPC codes, BP + OSD enables high-performance decoding of both topological QLDPC codes and our new class of semitopological codes. To this end, we first run numerical simulations of BP + OSD on the toric code with increasing code distances. Our results indicate a threshold in the region $9.9\% \pm 0.2\%$, in addition to showing evidence of exponential suppression in the low error regime. This BP + OSD threshold improves upon previous BP-based decoders for the toric code [38], and is close to the value of 10.3% achieved by state-of-the-art decoders for the toric code based on the minimum-weight perfect-matching algorithm [15,40,41]. We perform further numerical simulations of BP + OSD applied to a family of semitopological codes, as well as a family of finite-rate random QLDPC codes. For large block sizes, the BP + OSD threshold obtained for the semitopological codes approaches the value obtained for the toric code, reflecting the fact that the majority of stabilizer checks are 2D local.

This paper is structured as follows. In Sec. II, we first review essential concepts in classical coding theory, before introducing the edge-augmentation procedure. Section III covers the basics of quantum stabilizer codes, and explains how they can be represented as binary linear codes. In Sec. IV, we describe how QLDPC codes are obtained from classical LDPC codes via the hypergraph product, giving explicit examples of the construction of topological and random QLDPC codes. Following this, we explain how semitopological codes are constructed by taking the hypergraph product of augmented parent codes. In Sec. V we describe the workings of the BP + OSD decoder. In Sec. VI, we describe the "combination sweep" strategy as a greedy search method for finding higher-order solutions to BP + OSD. Following this, we present the results of our numerical simulations of the BP + OSD decoder for topological QLDPC codes, semitopological codes, and random QLDPC codes. Finally, in Sec. VII we summarize and discuss directions for future work.

II. LOW-DENSITY PARITY-CHECK CODES

Classical error correction. A classical error correction code $\mathcal{C}_H$ describes a redundant encoding $\mathbf{b} \mapsto \mathbf{c}$ from a k-bit data string $\mathbf{b}$ to an n-bit code word $\mathbf{c}$ (where $n > k$). The code words $\mathbf{c} \in \mathcal{C}_H$ are defined as the null space vectors of an $m \times n$ binary parity check matrix H such that $H \cdot \mathbf{c} \mod 2 = \mathbf{0}$.[1] By the rank-nullity theorem, a parity matrix permits

[1]From this point on, we assume all arithmetic is performed modulo 2.

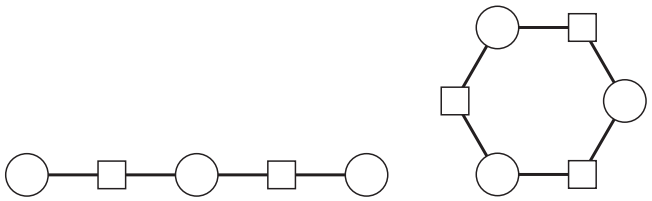

FIG. 1. Factor graphs for two instances of the three-bit repetition code. Data nodes are drawn as circles, parity nodes as squares, and edges as solid black lines. (a) The full-rank [3,1,3] repetition code with parity check matrix $H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$. (b) The closed-loop [3,1,3] repetition code (also known as the ring code) with parity check matrix $H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$.

$k = n - \text{RANK}(H)$ linearly independent code words. If a code word is subject to an error $\mathbf{e}$, the parity check matrix yields an m-bit syndrome $\mathbf{s} = H \cdot (\mathbf{c} + \mathbf{e}) = H \cdot \mathbf{e}$. The syndrome will be nonzero for all errors of Hamming weight less than the code distance $|\mathbf{e}| < d$. In general, classical codes are labeled with the $[n, k, d]$ notation, where n is the code word length, k is the number of encoded bits, and d is the code distance. The code rate is given by the ratio $R = k/n$.

Factor graphs. The factor graph of an $[n, k, d]$ classical code is a bipartite graph $G = (V, U, \Lambda)$ with an adjacency matrix given by the code's parity check matrix H [42]. For an $m \times n$ parity check matrix H, the two sets of nodes in G are defined as follows: (1) data nodes $V = \{v_j | j = 1, \ldots, n\}$ corresponding to the columns of H and taking the bit values of the error $\mathbf{e}$; (2) parity nodes $U = \{u_i | i = 1, \ldots, m\}$ corresponding to rows of H and taking the bit values of the syndrome $\mathbf{s} = H \cdot \mathbf{e}$. A graph edge $\lambda_{ij} \in \Lambda$ is drawn between a pair of nodes $\{v_j, u_i\}$ if $H_{ij} = 1$. Factor graphs serve as a useful visualization of the parity check matrix with applications in code design and decoding [11,43]. Diagrammatically, factor graphs are drawn with circles representing data nodes, squares representing parity nodes, and solid lines representing the edges. Figure 1 shows factor graphs for two instances of the three-bit repetition code.

LDPC codes. A family of (l, q)-LDPC codes is defined as a set of codes whose parity check matrices have column and row weights upper bounded by l and q, respectively. As first demonstrated by Gallager [8], it is possible to construct an (l, q)-LDPC code by randomly generating a parity check matrix with the desired column and row weights. An alternative to random LDPC code search is to employ graphical constructions in which an LDPC code family is obtained by systematically modifying the factor graph of a base code.

Edge-augmented LDPC codes. We now introduce "edge augmentation" as a graphical method for creating an LDPC code family from the starting point of any "parent" factor graph $G = (V, U, \Lambda)$. In Sec. III, we show how semitopological codes are created by taking the hypergraph product of such augmented codes.

Focusing first on a single edge λ_{ij} connecting nodes $\{v_j, u_i\}$ in the parent code, the edge-augmentation operation involves the addition of a "graph chain segment" $G^g = \{V^g, U^g, \Lambda^g\}$ containing g data nodes $V^g = \{v_j^g | j = 1, \ldots, g\}$ and g parity nodes $U^g = \{u_i^g | i = 1, \ldots, g\}$. The adjacency matrix H^g of the graph chain segment has dimensions $g \times g$. Its general form is obtained by taking a size-g identity matrix and adding a "1" to the right of each of the first $g - 1$ entries in the diagonal. As an example, the adjacency matrix of a graph chain segment with $g = 4$ is given by

$$H^{g=4} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1}$$

Following addition of the graph chain segment to the parent graph G, the updated factor graph G' is written

$$G' = (V \cup V^g, U \cup U^g, \Lambda \setminus \{\lambda_{ij}\} \cup \Lambda^g \cup \Lambda^w), \tag{2}$$

where $\Lambda \setminus \{\lambda_{ij}\}$ is the original parent edge set minus the edge that has been augmented. Two additional edges $\Lambda^w = \{\lambda_{1j}^g, \lambda_{ig}^g\}$ are added to connect the nodes $\{v_j, u_1^g\}$ and $\{v_g^g, u_i\}$. These edges "weld" the graph chain segment to the parent nodes $\{v_j, u_i\}$.

A g-augmented factor graph $G^{\star g} = (V^{\star g}, U^{\star g}, \Lambda^{\star g})$ is obtained by edge-augmenting each edge in a parent graph $G = (V, U, \Lambda)$ with a length-g graph chain segment. If G corresponds to an $[n, k, d]$ code with parity check matrix H, then the g-augmented graph $G^{\star g}$ corresponds to an $[n + g|\Lambda|, k, d']$ code with parity check matrix $H^{\star g}$, where $|\Lambda|$ is the number of edges in the parent graph. The augmented code distance depends upon the structure of H, but is lower bounded by $d' \geqslant (1 + g\mu)d$, where μ is the minimum degree over all data nodes V (for the proof of this lower bound see Appendix A). If the parent graph G is an (l, q)-LDPC code with $l, q \geqslant 2$, then the g-augmented graph $G^{\star g}$ will also be an (l, q)-LDPC code. A family of LDPC codes with increasing code distance can be obtained by augmenting a parent code with increasing values of the augmentation parameter g. The trade-off in the edge-augmentation procedure is a reduction in the code rate: if the parent graph has rate $R = k/n$, the augmented graph will have rate $R^{\star g} = \frac{R}{1+g|\Lambda|/n}$. Any increases in code distance due to edge augmentation must therefore be balanced against the respective increase in code overhead.

Figure 2 illustrates the first three levels of a (2,3)-LDPC code family starting from a [3,2,2] parent code with parity check matrix $H = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. The factor graph of the parent code G is shown in Fig. 2(a). Figure 2(b) shows the g-augmented graph $G^{\star 1}$ with $g = 1$ and code parameters [9,2,6]. Here, each edge in the parent graph G has been augmented with a length-1 graph chain segment, the nodes of which are colored red. Figure 2(c) is the g-augmented graph $G^{\star 2}$ corresponding to a code with parameters [15,2,10].

III. QUANTUM CODING

Quantum error correction. Quantum bits (qubits) are susceptible to a continuum of errors corresponding to rotations about the Bloch sphere. Fortunately, due to an effect known as the digitization of the error, quantum errors can be modeled in terms of the random occurrence of a discrete set of Pauli

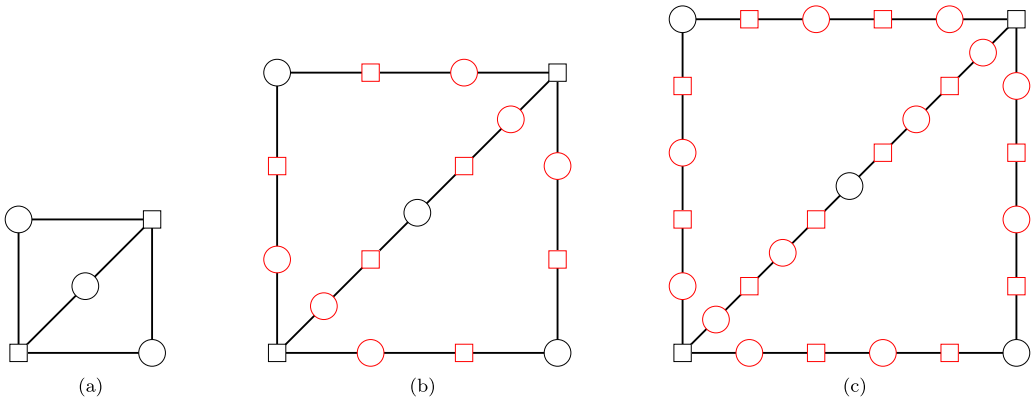

FIG. 2. Augmented (2,3)-LDPC codes. (a) The parent factor graph G with parity check matrix $H = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. (b) The g-augmented graph $G^{\star 1}$ with $g = 1$ corresponding to a [9,2,6] code. (c) The g-augmented code $G^{\star g}$ with $g = 2$ corresponding to a [15,2,10] code. The nodes belonging to the graph chain segments that form each augmented edge are colored red.

operators, $\{\mathbb{1}, X, Y, Z\}$,[2] on each qubit [44]. An $[[n, k, d]]$ quantum error correction code $\mathcal{Q}$ is a mapping $|\psi\rangle \mapsto |\psi\rangle_L$ from a k-qubit quantum state $|\psi\rangle$ to an entangled n-qubit code word (logical) state $|\psi\rangle_L$. The quantum code words $|\psi\rangle_L \in \mathcal{Q}$ satisfy the condition $S_j |\psi\rangle_L = (+1) |\psi\rangle_L$ for all $S_j \in \mathcal{S}$, where $\mathcal{S}$ is a group of mutually commuting Pauli operators known as the code's stabilizer [45]. Pauli errors of Hamming weight less than the code distance $|E| < d$ will result in at least one stabilizer S_k projecting onto the negative eigenspace $S_k |\psi\rangle_L = (-1) |\psi\rangle_L$.

The Pauli group has a convenient binary representation in which each operator is mapped to a length-2 vector: $\mathbb{1} \mapsto (0, 0)$, $X \mapsto (1, 0)$, $Z \mapsto (0, 1)$, and $Y \mapsto (1, 1)$. In general, the binary representation of an n-qubit Pauli operator K will be a length-$2n$ vector of the form $\mathbf{k} = (\mathbf{x}, \mathbf{z})$, where $\mathbf{x}$ and $\mathbf{z}$ both have length n and represent the positions of X- and Z-Pauli components respectively. As an example, the binary representation of the three-qubit Pauli operator $K = X_1 Z_3$ is $\mathbf{k} = (100, 001)$. The binary representation provides a useful setting from which to repurpose existing classical coding techniques for quantum error correction.

A quantum parity check matrix is defined as a matrix in which each row corresponds to a code stabilizer in its binary representation. Calderbank, Shor, and Steane (CSS) codes [46–48] are a subset of quantum codes with parity check matrices of the form $H_{\text{CSS}} = \begin{pmatrix} H_Z & 0 \\ 0 & H_X \end{pmatrix}$, where $H_Z \cdot H_X^T = \mathbf{0}$ due to the requirement that the stabilizers commute. For a CSS code subject to a Pauli error $E \mapsto \mathbf{e}_Q = (\mathbf{x}, \mathbf{z})$, the quantum syndrome $\mathbf{s}_Q$ is calculated as follows:

$$\mathbf{s}_Q = (\mathbf{s}_X, \mathbf{s}_Z) = (H_Z \cdot \mathbf{x},\ H_X \cdot \mathbf{z}). \tag{3}$$

From the above, it can be seen that the working of a CSS code can be thought of in terms of two classical codes, $\mathcal{C}(H_Z)$ and $\mathcal{C}(H_X)$, designed to detect bit flips (X errors) and phase flips (Z errors), respectively.

[2]The Pauli operators are defined as follows: $\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$; $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$; $Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}$; $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$.

Hypergraph product codes. The hypergraph product, first proposed by Tillich and Zemor [14], is a method for converting classical code pairs $\{\mathcal{C}_{H_1}, \mathcal{C}_{H_2}\}$ to a quantum CSS code $\mathcal{HGP}(\mathcal{C}_{H_1}, \mathcal{C}_{H_2})$. In the below, we describe the special case of the symmetric hypergraph product $\mathcal{HGP}(\mathcal{C}_H)$ for which $\mathcal{C}_{H_2} = \mathcal{C}_{H_1}$.

For a classical code $\mathcal{C}_H$ with code parameters $[n, k, d]$, the symmetric product $\mathcal{HGP}(\mathcal{C}_H)$ is a CSS code with

$$\begin{aligned} H_X &= (H \otimes \mathbb{1}_n | \mathbb{1}_m \otimes H^T), \\ H_Z &= (\mathbb{1}_n \otimes H | H^T \otimes \mathbb{1}_m), \end{aligned} \tag{4}$$

where H^T is the transpose parity check matrix describing a "transpose" code $\mathcal{C}_H^T$ with parameters $[m, k^T, d^T]$. Here, k^T is the number of logical qubits encoded by the transpose code while d^T is the distance of the transpose code. The quantum code parameters of $\mathcal{HGP}(H)$ are

$$[[n^2 + m^2,\ k^2 + (k^T)^2,\ \text{MIN}(d, d^T)]]. \tag{5}$$

The specific advantage of the hypergraph product construction is that it allows *any* classical code to be converted to a quantum code: the commutativity constraint $H_Z \cdot H_X^T = \mathbf{0}$ is satisfied for all binary parity check matrices H.

IV. QUANTUM LDPC CODES

An (l_Q, q_Q)-QLDPC code family is defined as a set of CSS codes whose quantum parity check matrices H_{CSS} have row and column weights upper bounded by l_Q and q_Q, respectively [49]. The hypergraph product preserves the sparsity of the original classical code [14]. From the structure of Eq. (4), we see that the hypergraph product of an (l, q)-LDPC code with parity check matrix H results in an (l_Q, q_Q)-QLDPC code with quantum parity check matrix H_Q, where $l_Q = \text{MAX}(2l, 2q)$ and $q_Q = l + q$. The hypergraph product of a classical LDPC code family is therefore a quantum LDPC (QLDPC) code family.

Two important classes of hypergraph product codes are as follows: (1) topological QLDPC codes, such as the surface and toric codes, constructed by taking the hypergraph product of repetition codes; (2) random QLDPC codes constructed by taking the hypergraph product of randomly generated

TABLE I. A constant-rate (8,7)-QLDPC code family $\mathcal{HGP}(\mathcal{C}_H)$ constructed from the hypergraph product of classical (3,4)-LDPC codes $\mathcal{C}_H$. Column 1: $[n, k, d]$ parameters of the classical (3,4)-LDPC codes $\mathcal{C}_H$. The parity check matrices of these codes have full rank. Column 2: $[m, k^T, d^T]$ parameters of the transpose codes $\mathcal{C}_H^T$. The distance of these codes is set to $d^T = \infty$ as they encode zero logical bits. Column 3: $[[n, k, d]]$ parameters of the (8,7)-QLDPC codes $\mathcal{HGP}(\mathcal{C}_H)$. Column 4: The rate of the QLDPC code $\mathcal{HGP}(\mathcal{C}_H)$. Column 5: the average check weight $\bar{w}$ of $\mathcal{HGP}(\mathcal{C}_H)$.

$\mathcal{C}_H$	$\mathcal{C}_H^T$	$\mathcal{HGP}(\mathcal{C}_H)$	$R = k/n$	$\bar{w}$
[16, 4, 6]	[12, 0, ∞]	[[400,16,6]]	0.04	7.0
[20, 5, 8]	[15, 0, ∞]	[[625,25,8]]	0.04	7.0
[24, 6, 10]	[18, 0, ∞]	[[900,36,10]]	0.04	7.0

classical LDPC codes. When random codes generate a factor graph with the expansion property, these are known as "quantum expander codes" [17–19]. In this section, we propose a new class of semitopological codes constructed by taking the hypergraph product of augmented LDPC code families. Semitopological codes are designed to share properties of both random and topological QLDPC codes.

Topological (4,4)-QLDPC codes. The hypergraph product of an $[n, 1, n]$ full-rank repetition code [see Fig. 1(a) for an example] yields a surface code with parameters $[[n^2 + (n-1)^2, 1, n]]$. Likewise, the hypergraph product of the closed-loop repetition code [also known as the ring code; see Fig. 1(b) for an example] results in a toric code with parameters $[[2n^2, 2, n]]$. Topological codes such as the surface code are considered leading candidates for experiment due to their high threshold [16] and the fact that they are local: all code stabilizers can be measured via interactions between nearest-neighbor qubits [26]. Another advantage of the topological codes is that they have parity check matrices that are (4,4)-QLDPC, meaning each stabilizer measurement involves at most four qubits. From a hardware perspective, this is beneficial, as each parity check operation involves error-prone multiqubit operations. The shortcoming of topological codes is that they scale poorly in terms of rate: $R = k/n \to 0$ as d is increased.

Random QLDPC codes. Random QLDPC codes are constructed from the hypergraph product of randomly generated classical LDPC codes [37]. The advantage of random QLDPC codes, over topological codes, is that they can encode more qubits per logical block. Table I lists members of an (8,7)-QLDPC code family constructed by taking the hypergraph product of a family of randomly generated (3,4)-LDPC codes. The (3,4)-LDPC classical code family was obtained using the Mackay-Neal method which ensures the randomly generated parity check matrix has no length-four cycles [13]. The resultant (8,7)-QLDPC hypergraph product codes are finite rate, with $R = k/n = 0.04$ as the distance is increased. The disadvantage of QLDPC codes is that they are highly nonlocal, requiring arbitrary qubit-qubit interconnectivity to perform stabilizer checks. Furthermore, the stabilizers typically involve more qubits than topological codes. The family of codes shown in Table I, for example, are (8,7)-QLDPC with stabilizer checks of mean weight $\bar{w} = 7.0$. This is higher than the mean check weight of $\bar{w} = 4.0$ for the (4,4)-QLDPC toric codes.

TABLE II. A semitopological code family $\mathcal{HGP}(\mathcal{C}_H^{*g})$ constructed from the augmented (2,3)-LDPC codes $\mathcal{C}_H^{*g}$. Column 1: the code augmentation parameter g. Column 2: $[n, k, d]$ parameters for the augmented (2,3)-LDPC codes. Column 3: $[m, k^T, d^T]$ parameters of the transpose code $(\mathcal{C}_H^{*g})^T$. Column 4: $[[n, k, d]]$ parameters of the semitopological code $\mathcal{HGP}(\mathcal{C}_H^{*g})$. These codes are (6,5)-QLDPC. Column 5: code rate $R = k/n$ of $\mathcal{HGP}(\mathcal{C}_H^{*g})$. Column 6: average check weight $\bar{w}$ of $\mathcal{HGP}(\mathcal{C}_H^{*g})$.

g	$\mathcal{C}_H^{*g}$	$(\mathcal{C}_H^{*g})^T$	$\mathcal{HGP}(\mathcal{C}_H^{*g})$	R	$\bar{w}$
0	[3,2,2]	[2,1,1]	[[13,5,2]]	0.385	5.00
1	[9,2,6]	[8,1,8]	[[145,5,6]]	0.0345	4.25
2	[15,2,10]	[14,1,14]	[[421,5,10]]	0.0119	4.14
3	[21,2,14]	[20,1,20]	[[841,5,14]]	0.00595	4.10
9	[57,2,38]	[56,1,56]	[[6385,5,38]]	0.000783	4.04

Semitopological codes. Semitopological codes are constructed by taking the hypergraph product of augmented LDPC codes. Table II shows the code parameters of a family of semitopological codes constructed from (2,3)-LDPC augmented codes of the type illustrated in Fig. 2. For an augmented code $\mathcal{C}_H^{*g}$, each augmented edge can be thought of as a section of a repetition code. The hypergraph product $\mathcal{HGP}(\mathcal{C}_H^{*g})$ therefore maps each augmented edge to a section of code that resembles a surface code. In these regions, the code stabilizers will be local. As the distance of the augmented code is increased, the resultant semitopological code contains larger surface code patches and becomes more local in nature. This convergence to surface-code-like structure is shown by the check-weight parameter $\bar{w}$ in Table II, which tends to 4.0 with increasing code distance as the local surface-code-like patches begin to dominate. We term this new family "semitopological codes," as they encode more logical qubits than the topological codes while requiring fewer long-range interactions than random QLDPC codes.

V. BELIEF PROPAGATION DECODING

In the classical setting, the role of the decoder is to determine the most likely error string $\mathbf{e}$ satisfying the syndrome equation $H \cdot \mathbf{e} = \mathbf{s}$. In practice, this decoding problem amounts to finding a minimum weight (MW) estimate of the error $\mathbf{e}_{\mathrm{MW}} \mapsto \mathrm{ARGMAX}_{\mathbf{e}} P(\mathbf{e}|\mathbf{s})$. For a uniformly distributed random noise model, the MW estimate can be computed bitwise by calculating the marginal probability that bit $e_i = 1$ as follows:

$$P_1(e_i) = \sum_{\sim e_i} P(e_1, e_2, e_i = 1, e_3, \ldots, e_n|\mathbf{s}), \tag{6}$$

where $\sum_{\sim e_i}$ denotes a summation over all bits e_j except e_i. The marginal $P_1(e_i)$ is referred to as a *soft decision* for the bit e_i. The final decoding estimate (*hard decision*) is then made for each bit according to

$$(e_{\mathrm{MW}})_i = \begin{cases} 1, & \text{if } P_1(e_i) \geqslant 0.5, \\ 0, & \text{if } P_1(e_i) < 0.5. \end{cases} \tag{7}$$

Belief propagation (BP) is an efficient marginalization algorithm and the backbone of many high-performance classical decoders [50]. The essential intuition underpinning BP is that

(for certain codes) the probability distribution $P(\mathbf{e}|\mathbf{s})$ can be factorized in a way that reduces the number of repeat summations in the computation of the marginals. The specific form of this factorization is deduced from the structure of the code's factor graph. The BP algorithm computes exact marginals when applied to codes with treelike factor graphs. For factor graphs with loops, the BP decoder outputs approximate marginals. However, it has been shown [13] that good decoding performance is nonetheless possible provided the factor graph is sufficiently loop-free.

The BP decoder takes a parity check matrix H and a syndrome $\mathbf{s}$ as input. The algorithm iteratively updates a soft-decision vector $P_1(\mathbf{e})$ by passing sets of "beliefs" between the nodes of the factor graph. At each iteration, a BP estimate $\mathbf{e}_{\text{BP}}$ is obtained via a hard decision on $P_1(\mathbf{e})$. If the BP estimate satisfies the syndrome equation, $H \cdot \mathbf{e}_{\text{BP}} = \mathbf{s}$, the BP decoder is said to have "converged" and the BP algorithm is terminated. The BP decoder fails if convergence does not occur within a number of iterations equal to the block length of the code. A more detailed description of BP can be found in Appendix C.

BP decoding of quantum codes. For a CSS code subject to a Pauli error $E \mapsto \mathbf{e}_Q = (\mathbf{x}, \mathbf{z})$, the quantum syndrome is given by $\mathbf{s}_Q = (\mathbf{s}_x, \mathbf{s}_z) = (H_Z \cdot \mathbf{x},\ H_X \cdot \mathbf{z})$. Assuming a Pauli-noise model with uncorrelated X and Z errors, the CSS code can be decoded independently as two classical codes with syndrome equations $\mathbf{s}_x = H_Z \cdot \mathbf{x}$ and $\mathbf{s}_z = H_X \cdot \mathbf{z}$. Unfortunately, the unmodified BP algorithm cannot be used to directly decode CSS codes owing (in part) to an effect known as quantum degeneracy. Quantum degeneracy arises due to the fact that there can be multiple minimum-weight solutions to the quantum decoding problem. In classical coding the goal is to estimate the exact error configuration that occurred $\mathbf{e}_{MW} = \mathbf{e}$. In contrast, for quantum coding, it is sufficient to find any recovery operation $\mathbf{r}_Q$ that is equivalent to the error up to a stabilizer $\mathbf{r}_Q + \mathbf{e}_Q = \text{ROWSPACE}(H_{\text{CSS}})$. For BP decoding, quantum degeneracy becomes problematic when there are multiple minimum-weight solutions satisfying the syndrome equation. As an example, consider a bit error decoding problem $\mathbf{s}_x = H_Z \cdot \mathbf{x}$ that has two minimum-weight solutions $\mathbf{x}_1$ and $\mathbf{x}_2$. As the degenerate solutions have equal Hamming weight $|\mathbf{x}_1| = |\mathbf{x}_2|$ the BP decoder assigns high probability to both. This situation is referred to as a split belief [15], and leads to a BP output of the form $\mathbf{x}_{\text{BP}} = \mathbf{x}_1 + \mathbf{x}_2$. In this case, $H_Z \cdot \mathbf{x}_{\text{BP}} = \mathbf{s}_x + \mathbf{s}_x = \mathbf{0} \neq \mathbf{s}_x$. The BP decoder therefore fails to converge when there are split beliefs of this type.

Ordered statistics decoding. Many attempts have been made to modify or supplement the BP algorithm to solve the problem of quantum degeneracy. The most successful approach to date involves applying a post-processing algorithm known as the ordered statistics decoder (OSD). Originally designed as a method for reducing error floors in classical LDPC codes by Fossorier and Lin [23], OSD was first applied in the quantum setting by Panteleev and Kalachev [22] and shown to be a surprisingly effective decoder of random QLDPC codes. In this paper, we show that OSD also performs well for the toric codes and our new class of semitopological codes. We also provide the first open-source demonstration of the algorithm [51]. Note that in the below, for notational simplicity, we describe OSD post-processing as applied to a classical decoding problem $\mathbf{s} = H \cdot \mathbf{e}$. The procedure we outline applies equally to decoding the H_X and H_Z components of a CSS code.

As parity check matrices do not have full column rank, it is not possible to solve the syndrome equation by matrix inversion $H^{-1} \cdot \mathbf{s} = \mathbf{e}$. However, for any parity check matrix it is possible to find a subset of columns, specified by the indices $[S]$, that are linearly independent. These columns form a basis and can be used to define a submatrix $H_{[S]}$ with full column rank, formed by selecting the columns $[S]$ of the original parity check matrix H. As this submatrix has full column rank, it can be inverted to give a solution to the syndrome equation $H_{[S]}^{-1} \cdot \mathbf{s} = \mathbf{e}_{[S]}$. Each choice of the basis $[S]$ corresponds to a unique solution $\mathbf{e}_{[S]}$, eliminating any potential ambiguity due to quantum degeneracy. It is possible to select $[S]$ as a random basis set, but this approach is unlikely to result in a good (low-weight) solution for $\mathbf{e}_{[S]}$. The idea behind the OSD post-processing algorithm is that the soft decisions from BP are used to select a basis set $[S]$ containing bits that have high probability of having been flipped.

The OSD-0 algorithm. In a BP + OSD decoder, the OSD post-processing step is called when the BP algorithm fails to converge within a number of iterations equal to the block length of the code. The simplest manifestation of the OSD decoder is known as OSD-0, the steps of which are as follows:

(1) Use the BP soft decision vector $P_1(\mathbf{e})$ to obtain a ranked list of bit indices $[O_{\text{BP}}]$ ordered (left to right) from most to least likely of being flipped.

(2) Order the columns of the parity check matrix $H_{[O_{\text{BP}}]}$ according to the ranking $[O_{\text{BP}}]$.

(3) Select the first $\text{RANK}(H)$ linearly independent columns of $H_{[O_{\text{BP}}]}$ as the most probable basis set $[S]$.

(4) Calculate the OSD-0 solution on the basis bits by matrix inversion $\mathbf{e}_{[S]} = H_{[S]}^{-1} \cdot \mathbf{s}$.

(5) The OSD-0 solution across all bits is given by $\mathbf{e}_{[S,T]} = (\mathbf{e}_{[S]}, \mathbf{e}_{[T]}) = (\mathbf{e}_{[S]}, \mathbf{0})$, where we define the remainder set $[T]$ as the bits which are not in the basis set $[T] \notin [S]$. The OSD-0 solution will always satisfy the syndrome equation $H_{[S,T]} \cdot \mathbf{e}_{[S,T]} = \mathbf{s}$.

(6) Map the OSD-0 solution to the original bit ordering $\mathbf{e}_{[S,T]} \mapsto \mathbf{e}_{\text{OSD-0}}$.

Higher-order OSD. In higher-order OSD, we consider solutions for which $\mathbf{e}_{[T]} \neq \mathbf{0}$. The first step involves computing the OSD-0 solution $\mathbf{e}_{[S]}$ on the basis bits as described above. Following this, for a given choice of $\mathbf{e}_{[T]}$, the higher-order OSD solution across all bits is given by

$$\mathbf{e}_{[S,T]} = \left(H_{[S]}^{-1} \cdot \mathbf{e}_{[S]} + H_{[S]}^{-1} \cdot H_{[T]} \cdot \mathbf{e}_{[T]},\ \mathbf{e}_{[T]}\right). \tag{8}$$

Note that the above solution satisfies the syndrome relation $H_{[S,T]} \cdot \mathbf{e}_{[S,T]} = \mathbf{s}$ for all possible configurations of $\mathbf{e}_{[T]}$. A higher-order OSD routine involves searching over different values of $\mathbf{e}_{[T]}$ to find the OSD solution with the lowest Hamming weight $\text{MIN}(|\mathbf{e}_{[S,T]}|)$. The length of the $\mathbf{e}_{[T]}$ vector is equal to $k' = n - \text{RANK}(H)$, meaning there are $2^{k'}$ distinct configurations: as a result, searching over all configurations soon becomes intractable for large codes. However, the BP soft-decision vector $P_1(\mathbf{e})$ can be used to rank the bits in $\mathbf{e}_{[T]}$. Good solutions can then be discovered by implementing a weighted greedy search routine which prioritizes the more probable configurations of $\mathbf{e}_{[T]}$ according to the soft decisions $P_1(\mathbf{e})$.

Greedy search strategies for higher-order OSD. For the numerical simulations in this paper, we implement a greedy search method we refer to as the "combination sweep strategy," a variant of the method originally proposed in [23]. The steps of the combination sweep strategy are as follows:

(1) The bits in the $\mathbf{e}_{[T]}$ component of the OSD solution are sorted according to the BP soft decisions.

(2) All weight-one configurations of $\mathbf{e}_{[T]}$ are searched over.

(3) All weight-two configurations in the first λ bits of $\mathbf{e}_{[T]}$ are searched over. The total number of configurations considered is equal to $k' + \binom{\lambda}{2}$, where $k' = n - \text{RANK}(H)$ is the length of the $\mathbf{e}_{[T]}$ vector.

We label our decoders using the combination sweep greedy search algorithm as BP + OSD-CS. For all the simulations in this work, we set the combination sweep search depth parameter to $\lambda = 60$. Note that in [22], Panteleev and Kalachev used a different greedy search method that involved testing all 2^λ permutations of the first λ bits in $\mathbf{x}_{[T]}$. For a fixed number of search terms, the combination sweep search algorithm provides a modest improvement in decoding performance over this exhaustive approach. For more details, see Appendix B.

VI. NUMERICAL SIMULATIONS

Simulation methodology for BP + OSD decoding. For the numerical simulations of the BP + OSD decoder in this work, we sample errors from the code capacity channel under the assumption that X- and Z-type errors are uncorrelated. As the quantum error correction codes we consider are constructed from a symmetric hypergraph product, the respective decoding problems for X- and Z-type errors are equivalent. As such, it suffices to simulate a single error species to assess decoding performance. Here, we sample X errors and solve the decoding problem $\mathbf{s}_x = H_Z \cdot \mathbf{x}$. The pseudocode for the specific implementation of BP we use for the numerical simulations in this paper can be found in Appendix C. The simulation chain we implement for each BP + OSD decoding cycle is described below:

(1) An error $\mathbf{x}$ is randomly sampled from a binary symmetric channel with bit error rate p. The syndrome is then calculated $\mathbf{s}_x = H_Z \cdot \mathbf{x}$.

(2) The BP decoder is called with H_Z and $\mathbf{s}$ as inputs. The output of the BP decoder is a candidate solution $\mathbf{x}_{\text{BP}}$ along with its respective soft-decision vector $P_1(\mathbf{x})$. If $H_Z \cdot \mathbf{x}_{\text{BP}} = \mathbf{s}_x$, then the BP decoder has converged and the simulation jumps directly to step 5. If $H_Z \cdot \mathbf{x}_{\text{BP}} \neq \mathbf{s}_x$, then the OSD post-processing routine (steps 3 and 4) is called. For our decoding simulations we use the "min-sum" variant of the BP algorithm as described in [52].

(3) The OSD-0 post-processing method, as described above, is used to obtain a solution of the form $\mathbf{x}_{[S,T]} = (\mathbf{x}_{[S]}, \mathbf{x}_{[T]}) = (\mathbf{x}_{[S]}, \mathbf{0})$.

(4) A greedy algorithm is run to search for higher-order OSD solutions that improve upon OSD-0. For this work, we adopt the combination sweep strategy with the search depth parameter set to $\lambda = 60$. However, in general, the specific form of the greedy search routine can be tailored according to parameters such as the physical error rate or code structure. The lowest weight OSD solution, $\text{MIN}|\mathbf{e}_{[S,T]}|$, is mapped to the original bit ordering and chosen as the BP + OSD candidate solution $\mathbf{e}_{\text{OSD}}$.

(5) After applying the recovery provided by the decoder, the "residual" error is given by $\mathbf{x}_R = \mathbf{x} + \mathbf{x}_{\text{OSD}}$ (or in the case where BP converged $\mathbf{x}_R = \mathbf{x} + \mathbf{x}_{\text{BP}}$). The decoding cycle is counted as a success if $\mathbf{x}_R$ is a not an X-type logical operator of the code. By definition, an X-type logical operator will anticommute with its corresponding Z-type logical operator. Checking for decoding success therefore involves verifying that $L_Z \cdot \mathbf{x}_R = \mathbf{0}$, where L_Z is a matrix in which each row represents a Z-type logical operator.

Next, we discuss our thresholds estimates for three code families across the QLDPC code spectrum, with an overview presented in Table III.

TABLE III. Observed thresholds for numerical simulations of the BP+OSD decoder applied to toric, semitopological, and random QLDPC codes.

Code	BP	BP+OSD-0	BP+OSD-CS
Toric	N/A	$9.2\% \pm 0.2\%$	$9.9\% \pm 0.2\%$
Semitopological	N/A	$9.1\% \pm 0.2\%$	$9.7\% \pm 0.2\%$
Random	$6.5\% \pm 0.1\%$	$6.7\% \pm 0.1\%$	$7.1\% \pm 0.1\%$

Topological QLDPC codes. Figure 3 shows a toric code threshold plot comparing the BP decoder against the BP + OSD-CS decoder. The logical error rate p_L is plotted against the physical error rate p for code distances $d = \{9, 11, 13, 15\}$. Due to quantum degeneracy, the BP decoder

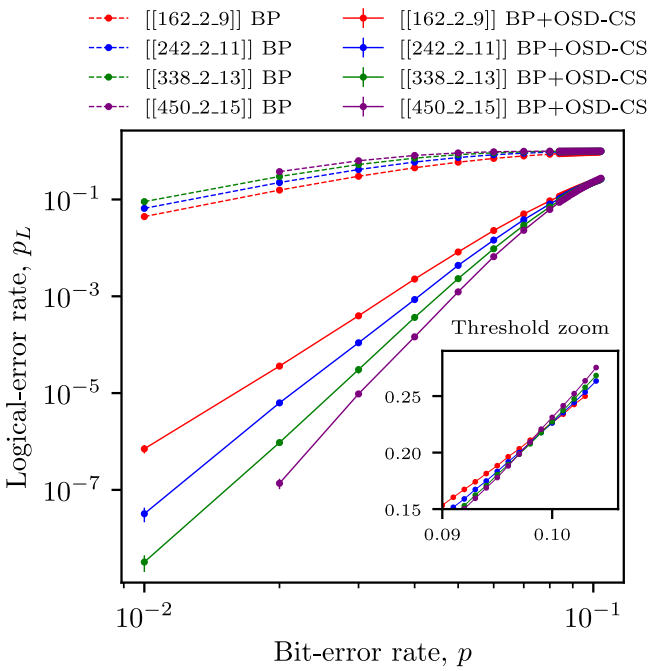

FIG. 3. Toric code threshold plot comparing the BP decoder (dashed lines) versus the BP + OSD-CS decoder (solid lines). The logical error rate p_L is plotted against the physical error rate p for code distances $d = \{9, 11, 13, 15\}$. For this simulation, the search depth parameter for the greedy search "combination sweep strategy" is set to $\lambda = 60$.

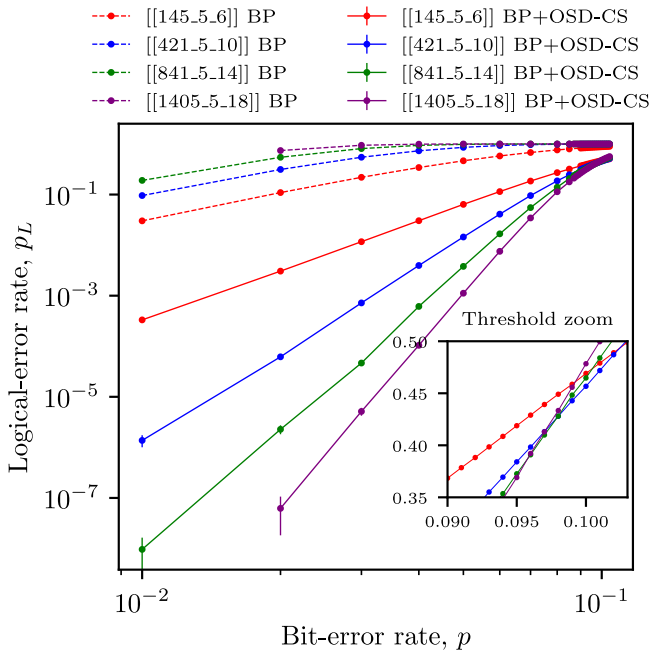

FIG. 4. Threshold plot for the semitopological codes constructed from a family of augmented codes (see Table II for the code parameters). The logical error rate p_L is plotted against the physical error rate p for code distances $d = \{6, 10, 14, 18\}$. The search depth parameter for the greedy search combination sweep strategy is set to $\lambda = 60$.

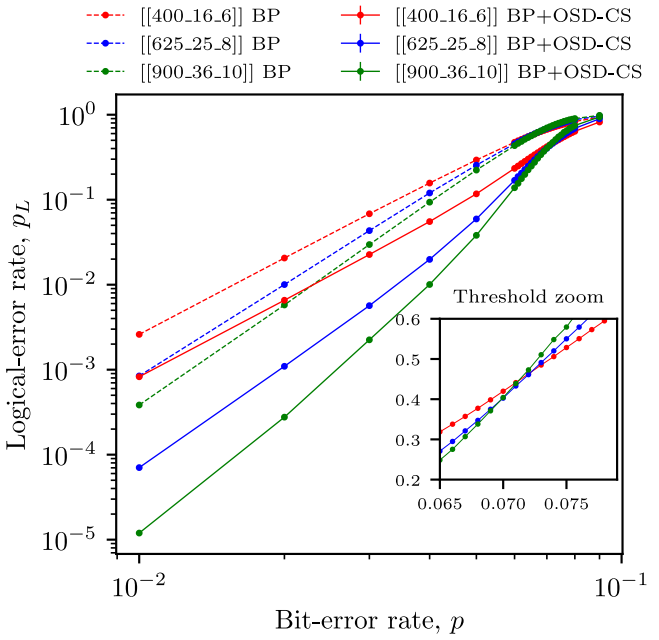

FIG. 5. Threshold plots for the family of constant-rate QLDPC codes listed in Table I. The logical error rate p_L is plotted against the physical error rate p for code distances $d = \{6, 8, 10\}$. The search depth parameter for the greedy search combination sweep strategy is set to $\lambda = 60$.

alone (dashed lines) does not exhibit a threshold: increasing the code distance d increases the logical error rate p_L for all values of the bit error rate p. In contrast, the BP + OSD-CS decoder (solid lines) shows crossings that indicate a threshold in the region $9.9\% \pm 0.2\%$. Furthermore, by inspection of the subthreshold regime, we see evidence of exponential suppression in the logical error rate with decreasing physical error rate. The corresponding threshold (not plotted) for the BP + OSD-0 decoder is $9.2\% \pm 0.2\%$. Performing the combination sweep for higher-order OSD solutions therefore results in a quantifiable improvement in decoding performance.

Semitopological codes. Figure 4 shows the threshold plot for a family of semitopological codes constructed from augmented (2,3)-LDPC codes. The parameters for this code family are listed in Table II. The logical error rates p_L (for both BP and BP + OSD-CS) are plotted against the physical error rate p for code distances $d = \{6, 10, 14, 18\}$. As with the toric codes, the BP decoder alone does not yield a threshold. For the BP + OSD-CS decoder, however, a crossing is clearly visible, suggesting a threshold in the range $9.7\% \pm 0.2\%$. Similarly to the toric code, inspection of the subthreshold regime shows evidence of exponential suppression for the semitopological code family. Within the margin of error, the semitopological code threshold aligns with the threshold for the toric code using the same decoder. This is the expected behavior, reflecting the fact that semitopological codes become structurally similar to toric codes (more local) as their distance is increased. The structural similarity arises because the chain segments are mapped to toric-code-like patches by the hypergraph product, and these regions form the bulk in the limit of large g. Discrepancies in the threshold between the semitopological codes and toric codes can be attributed to finite-size effects.

Random QLDPC codes. Figure 5 shows the results of numerical simulations of the BP + OSD decoder applied to the finite-rate family of random QLDPC codes summarized in Table I. The code distances considered are $d = \{6, 8, 10\}$. In contrast to the toric and semitopological codes, the BP decoder alone (before any OSD post-processing) shows a crossing, pointing to a threshold in the range $6.5\% \pm 0.1\%$. The existence of this threshold for the BP decoder can be attributed to the fact that random QLDPC codes are less structured than toric and semitopological codes; the repeating patterns present in stabilizer checks of topological codes lead to high densities of degenerate errors that cause BP to fail. The full BP + OSD-CS decoder applied to the random QLDPC family results in a threshold in the range $7.1\% \pm 0.1\%$. While this threshold value is only a modest improvement over BP, the real benefit of the OSD post-processing for random QLDPC codes becomes apparent in the low-error regime; at $p = 0.01$, for example, the logical error rate p_L for the BP + OSD-CS decoder is approximately an order magnitude less than that for BP.

VII. SUMMARY

Quantum LDPC codes have traditionally been studied as local topological codes or nonlocal random codes. In this paper we introduce semitopological codes as a means of interpolating on the local to nonlocal QLDPC spectrum.

Previously, the practicality of QLDPC codes has been hindered by the lack of a general purpose decoder: designing a new family of QLDPC codes would necessitate the development of a special-purpose decoding strategy [19,20]. In this paper, we provide further evidence that the recently proposed BP + OSD decoder [22] applies to all QLDPC codes constructed via the hypergraph product, including our new family of semitopological codes.

The methods for constructing semitopological codes proposed in this paper allow the locality of QLDPC codes to be balanced against other factors such as code rate. The existence of a general purpose BP + OSD decoder for QLDPC codes grants quantum computer architects more freedom in the design of fault-tolerant quantum computers; modifications to the structure of a QLDPC code can be made according to demands of a given device, without compromising the practicality of their decoding.

All of the simulations in this work were run under the assumption that the syndrome measurements are noiseless. In reality, syndrome extraction is performed using ancilla qubits with imperfect readout. In work currently in preparation [53], we study the performance of the BP + OSD decoder for higher-dimensional hypergraph product codes with the single-shot property [54–56], designed with in-built protection against syndrome noise.

Since our semitopological codes contain local patches of surface code, it would be useful to determine whether other QLDPC codes can be modified to contain such patches. For instance, Panteleev and Kalachev [22] constructed a $[[1270, 28, d]]$ (with unknown d) code that was especially competitive with surface codes. However, this code was constructed using a generalized hypergraph product and it is unclear whether an analog of our edge-augmentation process can be applied to this more general code family.

In this paper we have demonstrated the versatility of BP + OSD as a decoder across the spectrum of QLDPC codes that can be obtained from the hypergraph product. Beyond this, we conjecture that BP + OSD decoders will apply more generally to QLDPC codes constructed using different methods. Potential candidates for future investigation include topological fracton codes [57] and QLDPC codes based on high-performance classical protocols [58].

The code for the BP + OSD decoder used for the simulations in this paper can be downloaded from Github [51].

ACKNOWLEDGMENTS

J.R., S.B., and E.C. are supported by the QCDA project (EP/R043825/1) which has received funding from the QuantERA ERA-NET Cofund in Quantum Technologies implemented within the European Union's Horizon 2020 Programme. E.C. is additionally supported by the Engineering and Physical Sciences Research Council (EP/M024261/1). D.W. was supported by a research grant from Huawei. We thank Armanda Quintavalle for related discussions and comments throughout the project. The authors are grateful for the use of the following open source software packages: software for LDPC codes [59], Scipy [60], Numpy [61], and Matplotlib [62].

APPENDIX A: LOWER BOUND ON THE INCREASE IN CODE DISTANCE DUE TO EDGE AUGMENTATION

Theorem 1. Consider an $[n, k, d]$ classical code with Tanner graph $G = (V, U, \Lambda)$ and let μ denote the minimum degree over all data nodes V. Let $G^{\star g}$ be the Tanner graph resulting from augmenting each edge of G with g data nodes and g parity nodes. It follows that $G^{\star g}$ corresponds to an $[n + g|\Lambda|, k, d']$ code with $d' \geqslant (1 + g\mu)d$.

Proof. For the augmented graph $G^{\star g}$, we divide the data qubits $V \cup V^g$ into two disjoint subsets: the parent data nodes V and the augmented data nodes V^g. We let A denote a subset of data qubits $A \subseteq V \cup V^g$ that corresponds to a code word of the classical code, which is the case if and only if every check node in $G^{\star g}$ has an even number of graph neighbors in the set A. Furthermore, because the augmented data nodes V^g are all degree two, for each graph chain segment either all the data nodes are in A or none of them are. Furthermore, for every parent data node $a \in A \cap V$, it follows that every whole graph chain segment welded to a must be in A. Using #chains(A) to denote the number of graph chain segments present in A, we have that

$$|A| = |A \cap V| + g\#\text{chains}(A). \tag{A1}$$

Recall that each graph chain segment welds to one parent data node and one parent check node. Therefore, we can count the number of graph chain segments as follows: $\#\text{chains}(A) = \sum_{a \in A \cap V} \deg(a)$. In the graph $G^{\star g}$, the parent data nodes have the same degree as they did in the original graph G and by assumption this is lower bounded by μ. Therefore, we have $\#\text{chains}(A) \geqslant \mu|A \cap V|$ and so

$$|A| \geqslant (1 + g\mu)|A \cap V|. \tag{A2}$$

Next, we observe that A can only be a code word with respect to graph $G^{\star g}$ if $A \cap V$ is a code word with respect to graph G, which entails that

$$|A \cap V| \geqslant d. \tag{A3}$$

Let us break this observation down into steps. Assume to the contrary that $|A \cap V| < d$, so that with respect to graph G there is a parent check node $c \in U$ such that it has an odd number of neighbors in $A \cap V$. Furthermore, there will be an odd number of edges connecting c to $A \cap V$ in graph G. Each one of these edges maps to a graph chain segment in $A \cap V$ in the augmented graph, each of which welds to check node c. Therefore, check node c also has an odd number of neighbors with respect to graph $G^{\star g}$. This is impossible when A is a code word in graph $G^{\star g}$, so we must have that Eq. (A3) holds. Combining Eq. (A2) and Eq. (A3) gives $|A| \geqslant (1 + g\mu)d$ for any code word A in $G^{\star g}$, so this gives a lower bound on d'. ■

APPENDIX B: COMPARISON OF BP + OSD GREEDY SEARCH ALGORITHMS

The greedy search stage of a BP + OSD decoder involves testing different inputs to the OSD encoding operator given by

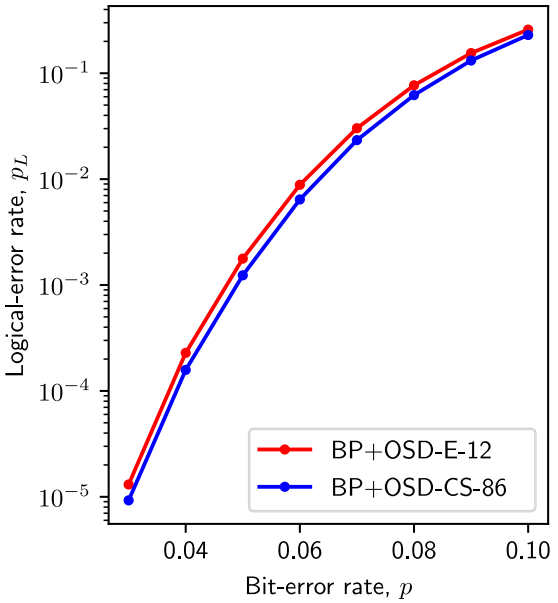

FIG. 6. Comparison of the BP + OSD-E and BP + OSD-CS methods when applied to the distance $d = 15$ toric code. The λ value for BP + OSD-E is set to $\lambda = 12$, leading to a total of 4096 inputs to the encoding operator defined in Eq. (8). For BP + OSD-CS, the λ value is set to $\lambda = 86$, leading to 3881 inputs to the encoding operator.

Eq. (8), with the aim of finding solutions that improve upon OSD-0. In [22], Panteleev and Kalachev use an exhaustive search strategy which we refer to as BP + OSD-E. For the simulations in this paper, we adopt the combination sweep method, referred to as BP + OSD-CS. For an OSD solution of the form, $\mathbf{e}_{\text{OSD}} = (\mathbf{e}_{[S]}, \mathbf{e}_{[T]})$, the first step is to order the bits in $\mathbf{e}_{[T]}$ according to the soft decisions from BP. The greedy search for the two methods then proceeds as follows:

(1) BP + OSD-E: All permutations of most-probable λ bits in $\mathbf{e}_{[T]}$ are searched over. In total, 2^λ search terms are considered.

(2) BP + OSD-CS: All weight one and two permutations in the first λ most-probable bits of $\mathbf{e}_{[T]}$ are searched over. The total number of configurations considered is equal to $k' + \binom{\lambda}{2}$, where $k' = n - \text{RANK}(H)$ is the length of the $\mathbf{e}_{[T]}$ vector.

Figure 6 compares decoding performance for the above OSD methods when applied to the $d = 15$ toric code. For the exhaustive method (BP + OSD-E), λ is set to $\lambda = 12$, giving a total of 4096 search terms. For the combination sweep method (BP + OSD-CS), λ is set to $\lambda = 86$, giving a total of 3881 search terms. Figure 6 shows that, despite considering fewer search terms, the BP + OSD-CS method improves decoding performance compared to BP + OSD-E.

APPENDIX C: MIN-SUM BELIEF PROPAGATION

We gave only a short summary of BP in the paper; we provide further details here to avoid the ambiguity often found in the literature, where BP is formulated for different applications and with many variations. We use a variant of the "min-sum" algorithm, using log-likelihood ratios for probabilities and the incorporation of variable scaling to prevent runaway values.

Belief propagation calculates marginal probabilities over graphical probabilistic models, a form of statistical inference, and is widely applied to the decoding of classical error-correcting codes. In the quantum domain the decoding task differs slightly in that, rather than trying to infer the original code word from the received message, we are given the syndrome indicating whether a given "parity check" failed and must infer a recovery operator; we must also cope with quantum degeneracy. Despite these differences, the task of quantum error correction (QEC) can be reformulated as a classical syndrome-based decoding problem. Unfortunately, syndrome-based decoding is not common in the classical decoding literature and there are few good references on the topic.

A more significant difference when applying BP to quantum codes is that all CSS and non-CSS QLDPC codes have factor graphs of girth four; originally BP was designed to work on acyclic graphs, but these factor graphs contain short cycles. While this violates the invariants of the algorithm and hence its proof of its correctness, empirically BP has been found to perform surprisingly well on cyclic graphs. Although it may sometimes fail to converge to a feasible solution, we can detect this by checking that its output satisfies the syndrome equation.

Formulating QEC decoding for BP

A QEC factor graph has data nodes representing each bit in the error string, which we denote v_j. It has one "check" or "parity" node for each syndrome measurement, which we denote u_i. The graph is described by the parity check matrix H (whether it concerns X or Z errors alone, or both, is immaterial; the methodology is the same). A one at position (i, j) in H indicates that parity node u_i has an edge directly connecting it to data node v_j.

There are two forms of prior information we must incorporate into the graph: the error rate of the channel, p, and the syndrome s. The error rate is incorporated as a hidden input to the data nodes. The syndrome measurement is implicitly present in the graph via calculations made at the parity nodes.

BP is conceptualized as a message-passing algorithm. We denote a message from a parity to a data node $m_{u_i \to v_j}$ and from a data node to a parity node as $m_{v_j \to u_i}$. As we possess only the syndrome, and not the received code word, the factor graph for QEC is slightly different from the standard graph found in classical decoding—but it is indeed equivalent to the (rarely discussed) syndrome-based classical decoding.

Our task is as follows: given the syndrome s and the structure represented by the factor graph, what is the most likely value of each bit in the error string?

Algorithm description

Algorithm 1. Pseudocode for belief propagation using log-likelihood ratios, the min-sum product algorithm, and a scaling factor. Log-likelihood ratios and the min-sum algorithm (the use of w in line 13) make the computation more efficient and avoid the numerical instability of other implementations.

1: **function** BELIEFPROP H, $\mathbf{s}$, p
2: ▷ Channel LLR
3: $p_l \leftarrow \log[(1-p)/p]$
4: ▷ (1) Initialization
5: **for** $(v_j, u_i) \in H$ **do**
6: $m_{v_j \to u_i} \leftarrow p_l$
7: **for** $iter \leftarrow 1$ to max **do**
8: ▷ Scaling factor
9: $\alpha \leftarrow 1 - 2^{-iter}$
10: ▷ (2) Parity to data messages
11: **for** $(u_i, v_j) \in H$ **do**
12: $w \leftarrow \min_{\sim v_j \in V(u_i)}\{|m_{v'_j \to u_i}|\}$
13: $m_{u_i \to v_j} = -1^{s_i}\alpha[\prod_{\sim v_j \in V(u_i)} \mathrm{sgn}(m_{v'_j \to u_i})]w$
14: ▷ (3) Data to parity messages
15: **for** $(v_j, u_i) \in H$ **do**
16: $m_{v_j \to u_i} \leftarrow m + \sum_{\sim u_i \in U(v_j)} m_{u'_i \to v_j}$
17: ▷ Hard decision
18: **for** $(v_j, u_i) \in H$ **do**
19: $P_1(e_j) \leftarrow p_j + \sum_{u'_i \in U(v_j)} m_{u'_i \to v_j}$
20: $\mathbf{e}^j_{\mathrm{BP}} \leftarrow -\mathrm{sgn}[P_1(e_j)]$
21: ▷ (4) Termination check
22: **if** $H \cdot \mathbf{e}_{\mathrm{BP}} = \mathbf{s}$ **then**
23: **return** True, $\mathbf{e}_{\mathrm{BP}}$, P_1
24: ▷ Failed to converge
25: **return** False, $\mathbf{e}_{\mathrm{BP}}$, P_1

The pseudocode for our implementation is given in Algorithm I, and consists of four sequential steps:

(1) *Initialization*. Messages are sent from data nodes to parity nodes giving the *a priori* probability of that bit in the error string being a one, i.e., the LLR (log-likelihood ratio) of the channel error rate p, which we denote p_l in its log-likelihood form:

$$p_l \triangleq \log[(1-p)/p]. \tag{C1}$$

(2) *Parity nodes to data nodes*. Messages are sent from parity nodes to data nodes containing the marginal probability of an error at the destination data node. However, we implement several optimizations that somewhat complicate the calculation of this message. Denoting the neighboring data nodes of a given parity node u_i as $V(u_i)$, the messages sent are

$$m_{u_i \to v_j} = -1^{s_i}\alpha\left[\prod_{v'_j \in V(u_i)\setminus v_j} \mathrm{sgn}\left(m_{v'_j \to u_i}\right)\right] \times \min_{v'_j \in V(u_i)\setminus v_j}\left\{\left|m_{v'_j \to u_i}\right|\right\}. \tag{C2}$$

The set minus notation in the subscripts indicates that this is a marginal distribution; i.e., we consider only the probabilities from other data nodes when calculating the marginal for this bit. The sign function and the first exponential $(-1)^{s_i}$ are used to incorporate the syndrome, with s_i being the ith bit of the syndrome. In other words: "consider all configurations of connected error bits, and increase the probability of the implied value for this bit compatible with the observed syndrome." The first factor is an XOR operation that establishes the sign of this probability, i.e., whether u_i is implied to be a one or a zero, based on the decision represented by the messages sent by other data bits. The second factor describes the magnitude of the probability and is based on the notion that the "cheapest" way that this value of u_i could be incorrect is if one of the other bits was flipped. For a full explanation, see [11].

We also include $alpha$, a scaling factor as outlined in [52]. The scaling factor α is set according to the current iteration $iter$, where the first iteration is numbered $iter = 1$:

$$\alpha = 1 - 2^{-iter}. \tag{C3}$$

(3) *Data nodes to parity nodes*. Next, messages are sent from data nodes to parity nodes giving the probability ratio for that bit in the error string, calculated by summing the inbound marginals and taking into account the error rate for the channel, omitting normalization for efficiency:

$$m_{v_j \to u_i} = p_l + \sum_{u'_i \in U(v_j)\setminus u_i} m_{u'_i \to v_j}, \tag{C4}$$

where we have denoted the neighboring data nodes of a given check node v_j as $U(v_j)$.

(4) *Termination check*. If the factor graph is a tree, we can always terminate after a single iteration of the algorithm. If it is cyclic (as in QEC), then we will terminate on success or else when a given number of iterations are complete. We first calculate a "hard decision" of the most likely error string, by selecting the most likely configuration via the bitwise marginals we have calculated:

$$P_1(e_j) = p_l + \sum_{u'_i \in U(v_j)} m_{u'_i \to v_j}. \tag{C5}$$

We then select the most likely error string $\tilde{E}$ given these bitwise probabilities, and calculate the expected syndrome:

$$\mathbf{s} = H \cdot \mathbf{e}. \tag{C6}$$

We terminate if $\mathbf{s}$ matches the measured syndrome, or if we have reached a preset maximum number of iterations (often equal to the block length). Otherwise, we return to step 2, sending "parity nodes to data nodes" messages.

The outputs of BP are both the soft and hard decisions; the former are used by OSD if BP has failed to converge, i.e., the hard decision does not satisfy the syndrome equation. The soft decision is a bitwise estimate of the probability an error occurred, which OSD uses to bias its search for an error string.

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

[2] D. Gottesman, An introduction to quantum error correction and fault-tolerant quantum computation, arXiv:0904.2557.

[3] S. J. Devitt, W. J. Munro, and K. Nemoto, Quantum error correction for beginners, Rep. Prog. Phys. **76**, 076001 (2013).

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

[5] J. Roffe, Quantum error correction: An introductory guide, Contemp. Phys. **60**, 226 (2019).

[6] N. Delfosse, Hierarchical decoding to reduce hardware requirements for quantum computing, arXiv:2001.11427.

[7] Z. Babar, P. Botsinis, D. Alanis, S. X. Ng, and Lajos Hanzo, Fifteen years of quantum LDPC coding and improved decoding strategies, IEEE Access **3**, 2492 (2015).

[8] R. Gallager, Low-density parity-check codes, IRE Trans. Inf. Theory **8**, 21 (1962).

[9] J. H. Bae, A. Abotabl, H.-P. Lin, K.-B. Song, and J. Lee, An overview of channel coding for 5G NR cellular communications, APSIPA Transactions on Signal and Information Processing **8**, E17 (2019).

[10] J. Pearl, Reverend Bayes on inference engines: A distributed hierarchical approach, in *Proceedings of the National Conference on Artificial Intelligence*, Carnegie Mellon University and the University of Pittsburgh, Pittsburgh, Pennsylvania (AAAI, 1982), pp. 133–136.

[11] F. R. Kschischang, B. J. Frey, H.-A. Loeliger *et al.*, Factor graphs and the sum-product algorithm, IEEE Trans. Inf. Theory **47**, 498 (2001).

[12] C. E. Shannon, A mathematical theory of communication, Bell System Tech. J **27**, 379 (1948).

[13] D. J. C. MacKay and R. M. Neal, Near Shannon limit performance of low density parity check codes, Electron. Lett. **33**, 457 (1997).

[14] J.-P. Tillich and G. Zémor, Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength, IEEE Trans. Inf. Theory **60**, 1193 (2013).

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

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

[17] A. Leverrier, J.-P. Tillich, and G. Zémor, Quantum expander codes, in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (IEEE, 2015), pp. 810–824.

[18] O. Fawzi, A. Grospellier, and A. Leverrier, Constant overhead quantum fault-tolerance with quantum expander codes, in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)* (IEEE, 2018), pp. 743–754.

[19] O. Fawzi, A. Grospellier, and A. Leverrier, Efficient decoding of random errors for quantum expander codes, in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing* (ACM, 2018), pp. 521–534.

[20] A. Grospellier and A. Krishna, Numerical study of hypergraph product codes, arXiv:1810.03681.

[21] A. Grospellier, Lucien Grouès, A. Krishna, and A. Leverrier, Combining hard and soft decoders for hypergraph product codes, arXiv:2004.11199.

[22] P. Panteleev and G. Kalachev, Degenerate quantum LDPC codes with good finite length performance, arXiv:1904.02703.

[23] M. P. C. Fossorier and S. Lin, Soft-decision decoding of linear block codes based on ordered statistics, IEEE Trans. Inf. Theory **41**, 1379 (1995).

[24] M. P. C. Fossorier, Iterative reliability-based decoding of low-density parity check codes, IEEE J. Select. Areas Commun. **19**, 908 (2001).

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

[26] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A **86**, 032324 (2012).

[27] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, E. Lucero, M. Neeley, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, and J. M. Martinis, Scalable *in situ* qubit calibration during repetitive error detection, Phys. Rev. A **94**, 032321 (2016).

[28] E. A. Sete, W. J. Zeng, and C. T. Rigetti, A functional architecture for scalable quantum computing, in *2016 IEEE International Conference on Rebooting Computing (ICRC)* (IEEE, 2016), pp. 1–6.

[29] M. Takita, A. W. Cross, A. D. Córcoles, J. M. Chow, and J. M. Gambetta, Experimental Demonstration of Fault-Tolerant State Preparation with Superconducting Qubits, Phys. Rev. Lett. **119**, 180501 (2017).

[30] J. Randall, S. Weidt, E. D. Standing, K. Lake, S. C. Webster, D. F. Murgia, T. Navickas, K. Roth, and W. K. Hensinger, Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions, Phys. Rev. A **91**, 012322 (2015).

[31] 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 (London) **536**, 63 (2016).

[32] M. F. Brandl, M. W. van Mourik, L. Postler, A. Nolf, K. Lakhmanskiy, R. R. Paiva, S. Möller, N. Daniilidis, H. Häffner, V. Kaushal, T. Ruster, C. Warschburger, H. Kaufmann, U. G. Poschinger, F. Schmidt-Kaler, P. Schindler, T. Monz, and R. Blatt, Cryogenic setup for trapped ion quantum computing, Rev. Sci. Instrum. **87**, 113103 (2016).

[33] C. J. Ballance, T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas, High-Fidelity Quantum Logic Gates Using Trapped-Ion Hyperfine Qubits, Phys. Rev. Lett. **117**, 060504 (2016).

[34] X. Qiang, X. Zhou, J. Wang, Callum M. Wilkes, T. Loke, Sean O'Gara, L. Kling, Graham D. Marshall, R. Santagati, Timothy C. Ralph, Jingbo B. Wang, Jeremy L. O'Brien, Mark G. Thompson, and J. C. F. Matthews, Large-scale silicon quantum photonics implementing arbitrary two-qubit processing, Nat. Photonics **12**, 534 (2018).

[35] X.-L. Wang, L.-K. Chen, W. Li, H.-L. Huang, C. Liu, C. Chen, Y.-H. Luo, Z.-E. Su, D. Wu, Z.-D. Li, H. Lu, Y. Hu, X. Jiang, C.-Z. Peng, L. Li, N.-L. Liu, Yu.-A. Chen, C.-Y. Lu, and J.-W. Pan, Experimental Ten-Photon Entanglement, Phys. Rev. Lett. **117**, 210502 (2016).

[36] Y. Wu, Y. Wang, X. Qin, X. Rong, and J. Du, A programmable two-qubit solid-state quantum processor under ambient conditions, npj Quantum Inf. **5**, 1 (2019).

[37] A. A. Kovalev, S. Prabhakar, I. Dumer, L. P. Pryadko, Numerical and analytical bounds on threshold error rates for hypergraph-product codes, Phys. Rev. A **97**, 062320 (2018).

[38] Y.-H. Liu and D. Poulin, Neural Belief-Propagation Decoders for Quantum Error-Correcting Codes, Phys. Rev. Lett. **122**, 200501 (2019).

[39] N. H. Nickerson, J. F. Fitzsimons, and S. C. Benjamin, Freely Scalable Quantum Technologies Using Cells of 5-to-50 Qubits with Very Lossy and Noisy Photonic Links, Phys. Rev. X **4**, 041041 (2014).

[40] J. Edmonds, Paths, trees, and flowers, Can. J. Math. **17**, 449 (1965).

[41] V. Kolmogorov, Blossom V: A new implementation of a minimum cost perfect matching algorithm, Math. Program. Comput. **1**, 43 (2009).

[42] R. Tanner, A recursive approach to low complexity codes, IEEE Trans. Inf. Theory **27**, 533 (1981).

[43] J. Roffe, S. Zohren, D. Horsman, and N. Chancellor, Quantum codes from classical graphical models, IEEE Trans. Inf. Theory **66**, 130 (2020).

[44] Emanuel Knill and R. Laflamme, Theory of quantum error-correcting codes, Phys. Rev. A **55**, 900 (1997).

[45] D. Gottesman, The Heisenberg representation of quantum computers, in *Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics* (International Press, Cambridge, MA, 1999), pp. 32–43.

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

[47] A. M. Steane, Error Correcting Codes in Quantum Theory, Phys. Rev. Lett. **77**, 793 (1996).

[48] A. M. Steane, Active Stabilization, Quantum Computation, and Quantum State Synthesis, Phys. Rev. Lett. **78**, 2252 (1997).

[49] D. J. C. MacKay, G. Mitchison, and P. L McFadden, Sparse-graph codes for quantum error correction, IEEE Trans. Inf. Theory **50**, 2315 (2004).

[50] D. J. C. MacKay, Good error-correcting codes based on very sparse matrices, IEEE Trans. Inf. Theory **45**, 399 (1999).

[51] J. Roffe, BP + OSD: Belief propagation with ordered statistics postprocessing for decoding quantum LDPC codes, https://github.com/quantumgizmos/bp_osd.

[52] A. A. Emran and M. Elsabrouty, Simplified variable-scaled min-sum LDPC decoder for irregular LDPC codes, in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)* (IEEE, 2014), pp. 518–523.

[53] A. O. Quintavalle, M. Vasmer, J. Roffe, and E. T. Campbell, Single-shot error correction of three-dimensional homological product codes, arXiv:2009.11790.

[54] H. Bombín, Single-Shot Fault-Tolerant Quantum Error Correction, Phys. Rev. X **5**, 031043 (2015).

[55] M. Vasmer and D. E. Browne, Three-dimensional surface codes: Transversal gates and fault-tolerant architectures, Phys. Rev. A **100**, 012312 (2019).

[56] E. Campbell, A theory of single-shot error correction for adversarial noise, Quantum Sci. Technol. **4**, 025006 (2019).

[57] S. Vijay, J. Haah, and L. Fu, Fracton topological order, generalized lattice gauge theory, and duality, Phys. Rev. B **94**, 235157 (2016).

[58] M. Hagiwara, K. Kasai, H. Imai, and K. Sakaniwa, Spatially coupled quasi-cyclic quantum LDPC codes, in *2011 IEEE International Symposium on Information Theory Proceedings* (IEEE, 2011), pp. 638–642.

[59] R. M. Neal, Software for low density parity check codes, http://radfordneal.github.io/LDPC-codes.

[60] P. Virtanen *et al.*, SciPy 1.0: Fundamental algorithms for scientific computing in Python, Nat. Methods **17**, 261 (2020).

[61] S. van der Walt, S. C. Colbert, and G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Comput. Sci. Eng. **13**, 22 (2011).

[62] J. D. Hunter, Matplotlib: A 2D graphics environment, Comput. Sci. Eng. **9**, 90 (2007).