

Toward Low-latency Iterative Decoding of QLDPC Codes Under Circuit-Level Noise

Anqi Gong*, Sebastian Cammerer†, and Joseph M. Renes*

*ETH Zürich, Switzerland, gonga@student.ethz.ch

†NVIDIA

Abstract—We introduce a sliding window decoder based on belief propagation (BP) with guided decimation for the purposes of decoding quantum low-density parity-check codes in the presence of circuit-level noise. Windowed decoding keeps the decoding complexity reasonable when, as is typically the case, repeated rounds of syndrome extraction are required to decode. Within each window, we employ several rounds of BP with decimation of the variable node that we expect to be the most likely to flip in each round. Furthermore, we employ ensemble decoding to keep both decimation options (guesses) open in a small number of chosen rounds. We term the resulting decoder BP with guided decimation guessing (GDG). Applied to bivariate bicycle codes, GDG achieves a similar logical error rate as BP with an additional OSD post-processing stage (BP+OSD) and combination-sweep of order 10. For a window size of three syndrome cycles, a multi-threaded CPU implementation of GDG achieves a worst-case decoding latency of 3ms per window for the $[[144,12,12]]$ code. The source code of this work is available online.¹

I. INTRODUCTION

The recent progress in asymptotically good quantum low-density parity-check (QLDPC) codes [1]–[4] renders them a promising candidate for low-overhead fault-tolerant quantum computing. Besides large blocklength codes, numerous short to medium blocklength ($N \lesssim 1000$) QLDPC codes with distance near—or even exceeding—the square root of blocklength have been proposed [5]–[7]. Their parity-check matrices are carefully designed, and some of them have certain structures that are promising for future use [8]–[10]. In particular, the bivariate bicycle (BB) codes [7] are numerically shown to be more resource-efficient than the planar surface codes [11].

Due to the noise in the syndrome measurement (SM) operations, the SM circuit is typically repeated for multiple cycles. This comes at the price of a considerable increase in decoding complexity. In this work, we employ a sliding window decoder based on belief propagation guided decimation (BPGD) [12] to handle streaming SM input.

The basic idea of window decoding is to use the syndrome outputs of a small number of subsequent rounds (the window) to determine the location of faults in the early part of the window, and then slide the window forward by a few rounds and repeat the process. An inner decoder is used on each window which ideally finds the lowest-weight correction such that the overall performance degradation compared to global decoding remains tolerable. The advantage of this approach is a lower decoding latency; no need to wait for a lengthy

decoding process that only begins after collecting the entire set of syndrome data.

Sliding window decoders have been applied to surface codes under circuit-level noise [11], [13], [14], where both the SM circuit and measurements are assumed to be noisy. Recently, it has also been applied to some QLDPC codes under phenomenological noise [15] where only the measurements are assumed to be noisy. In [15], belief propagation and ordered-statistics decoding (BP+OSD) [5], [16] is used on each window to achieve the aforementioned low-weight requirement.

We propose to use an extended version of the BPGD [12] algorithm as the inner decoder. BPGD interleaves BP runs with a *global* variable node (VN) selection and subsequent decimation of that VN. Compared to BP+OSD, the advantage is a possibly lower worst-case runtime, as BPGD avoids the need for Gaussian elimination. Furthermore, BPGD has demonstrated performance comparable to BP+OSD on data qubit noise decoding [12].

When applied to decoding circuit-level noise, however, we find that the BPGD VN selection rule, which is to choose the most reliable VN, becomes less efficient, since in this setting there are significantly more possible fault locations than actual fault occurrences. This motivates us to choose the most likely VN to flip at each step, with the idea that this will lead to BP convergence in fewer steps. Moreover, we use a short *history* of posterior log-likelihood ratios (LLRs) from recent BP iterations to choose the decimation value for the selected VN. Thereby, we build up a main decimation path of low depth. Lastly, we explore different decimation values (guesses) along the main path and at very early steps, see Fig. 2. These paths greatly improve the BP convergence speed and can run in independent parallel threads. We call our modified version of BPGD a guided decimation guessing (GDG) decoder. Note that, despite its name, GDG is a *deterministic* decoder since both options are explored, instead of a randomly chosen value for the chosen VN. We carefully optimize the maximum number of BP iterations and keep the number of paths small for GDG to be useful in the real-time setting.

The remainder of the paper is organized as follows. BP and BP+OSD decoding methods are reviewed in Section II-A, followed by a review of the global decoding of circuit-level noise [7] in Section II-B. Details of sliding window decoding are given in Section III and of the GDG decoder in Section IV. Additionally, in Appendix A and B, we apply both BP+OSD and GDG to data qubit noise and single-shot syndrome noise decoding to form comparisons.

¹<https://github.com/gongaa/SlidingWindowDecoder>

II. CIRCUIT-LEVEL NOISE DECODING

We follow the standard circuit-based depolarizing noise model [17], which is also considered in [7]. Each operation in the (repeated) syndrome measurement circuit, including CNOT gates, qubit initializations, measurements, and idle qubits, is subject to noise. One can imagine the gates occurring at integer timesteps starting at 1; then the possible fault locations are half-integer timesteps for all qubits (i.e. between all gates), including ancillas. The decoding on the much simpler data qubit error model is discussed in Appendix A, where it is assumed that the syndrome extraction circuit is noiseless and faults only occur at timestep 0.5.

The circuit-level noise decoder outputs a Pauli correction operator, to be applied to the output data qubits, upon observing all the noisy syndromes from multiple rounds and taking into account all possible faults in the SM circuit. The decoding fails if the correction operator and the actual error differ by a non-trivial logical operator of the code. Here we consider decoding using syndromes from X-type or Z-type measurements separately. This is enabled by the CSS structure [18], [19] of BB codes.

A. Decoding

Consider the Tanner graph [20] associated with a binary linear code specified by a parity-check matrix (PCM) $\mathbf{H}$. The Tanner graph is bipartite and consists of check nodes (CNs) and variable nodes (VNs) that each represent a row or a column of $\mathbf{H}$, respectively. The presence of one at column i and row j of $\mathbf{H}$ indicates an edge from VN v_i to CN c_j . Associated to this edge are messages in both directions, $\mu_{i \rightarrow j}$ and $\mu_{j \rightarrow i}$. In the following, we use the concept of VN and column interchangeably. A binary variable can be associated to each VN, indicating a fault on the corresponding qubit or locations in the data qubit noise and circuit-level noise scenarios, respectively.

In syndrome BP decoding, each CN c_j receives a syndrome (a check value) $s_j \in \{0, 1\}$, and we denote the entire vectors of syndromes $\mathbf{s}$. Each VN v_i receives as input a prior probability p_i of it being flipped, with associated log-likelihood ratio (LLR) $\Lambda_i = \log \frac{1-p_i}{p_i}$. We assume errors on VNs to be independent. For BP initialization (timestep $t = 0$), the VN v_i to CN c_j messages is $\mu_{i \rightarrow j}^{(0)} = \Lambda_i$.

The message-passing algorithm proceeds iteratively, with one timestep consisting of first CN updates and then VN updates. At timestep t , the min-sum CN update rule computes, for each CN c_j , a message $\mu_{j \rightarrow i}$ to each of its neighboring VNs v_i , with $i \in \mathcal{N}(j)$, where

$$\mu_{j \rightarrow i}^{(t)} = (-1)^{s_j} \cdot \min_{i' \in \mathcal{N}(j) \setminus i} |\mu_{i' \rightarrow j}^{(t-1)}| \prod_{i' \in \mathcal{N}(j) \setminus i} \text{sign}(\mu_{i' \rightarrow j}^{(t-1)}). \quad (1)$$

The VN update first calculates a posterior LLR $\Lambda_i^{(t)}$ for VN v_i at timestep t by summing up the original LLR Λ_i with all incoming messages from its CN neighbors $\mathcal{M}(i)$, i.e.,

$$\Lambda_i^{(t)} = \Lambda_i + \sum_{j' \in \mathcal{M}(i)} \mu_{i \leftarrow j'}^{(t)}. \quad (2)$$

Then the message $\mu_{i \rightarrow j}^{(t)}$ is updated by subtracting the intrinsic message from the posterior, i.e.,

$$\mu_{i \rightarrow j}^{(t)} = \Lambda_i^{(t)} - \mu_{i \leftarrow j}^{(t)}. \quad (3)$$

Based on the posterior LLR at any timestep t , a decision for the estimated error $\hat{e}$ can be made *locally* for each VN i

$$\hat{e}_i = \begin{cases} 0 & \text{if } \Lambda_i^{(t)} > 0 \\ 1 & \text{if } \Lambda_i^{(t)} \leq 0. \end{cases} \quad (4)$$

When the Tanner graph contains loops, the posterior LLR is only an approximated version of the true value; thus, it is possible that the BP's decision does not have the correct syndrome even after an infinite number of decoding iterations. Therefore, we stop BP as soon as $\hat{e}$ satisfies the syndrome equation $\mathbf{H}\hat{e} = \mathbf{s}$, or after some fixed number of iterations.

Since the prior LLRs on VNs need not be identical, which will be especially relevant in circuit-level decoding, we use the handy notation of the *path metric* (PM). It is defined to be the sum of prior LLRs from the VNs that are estimated to one, if the estimation has the correct syndrome, or ∞ otherwise.

$$\text{PM}(\hat{e}) = \begin{cases} \sum_{i: \hat{e}_i = 1} \Lambda_i & \text{if } \mathbf{H}\hat{e} = \mathbf{s}, \\ \infty & \text{if } \mathbf{H}\hat{e} \neq \mathbf{s}. \end{cases} \quad (5)$$

A smaller path metric, therefore, corresponds to a higher probability of the estimated error pattern.

In the later section, we will use a technique called VN decimation, which is to fix the value of a VN and remove it from the message-passing network. For example, if we choose to decimate v_i to 0, then this VN is no longer active and will be excluded from the updates of its CN neighbors. If we choose to decimate v_i to one, the syndrome s_j on all its CN neighbors $j \in \mathcal{M}(i)$ needs to be flipped, and the updated syndromes are used in Eq. (1) for later CN updates. In our implementation, instead of deleting a VN, we maintain a mask for the VN status. The messages from and to a decimated VN remain in the network, and all its CN neighbors $\mathcal{M}(i)$ ignore these *stale* messages in their updates.

In data qubit noise decoding of quantum LDPC codes, BP often outputs an error estimate $\hat{e}$ which does not satisfy the syndrome equation, this is usually termed non-convergence² and exhibits an error floor in logical error rates. One reason is due to the short loops present in the Tanner graph. The short to medium block-length QLDPC codes usually have a girth (the length of the shortest cycle) of four or six³.

A few classical tricks for loopy BP can be employed to ameliorate this problem. A normalization/scaling factor $\alpha \leq 1$ can be multiplied to the right-hand side of Eq. (1) to prevent over-amplified messages due to short cycles. The scaling

²Convergence and syndrome consistency are two different notions. The former does not imply the latter. However, we force the latter to imply the former by early stopping of BP once the syndrome equation is satisfied, in the sense that the estimation based on posterior LLR no longer changes.

³Here we consider the PCMs $\mathbf{H}_X$ and $\mathbf{H}_Z$ for X and Z errors separately. See Table (1) of [5] for the girth of some QLDPC codes. For quaternary BP decoding on $(\mathbf{H}_X, \mathbf{H}_Z)$, four-cycles are unavoidable [21].

factor is set to 1.0 in this work unless specified otherwise. Another trick is to change the scheduling method. Instead of all CNs applying the update rule in parallel in one BP iteration (flooding scheduling), only one CN (serial scheduling) or a fraction of CNs (layered scheduling) is updated before the next VN update. Serial scheduling usually reduces syndrome inconsistency with the same number of CN updates; however, those updates are sequential, which affects latency. In this work, we always use flooding scheduling within window decoding due to the potential latency constraint. Additionally, the (normalized) min-sum rule is used for the CN update, which is more hardware-efficient despite being an approximation.

Apart from short loops, symmetric trapping sets [22], [23] also complicate the decoding of QLDPC codes. Various post-processing methods have been proposed to break this symmetry, the most prominent being ordered-statistics decoding (OSD) [5], [16]. If BP fails to converge after some fixed number of iterations, the columns of $\mathbf{H}$ are reordered according to the posterior LLRs of the VNs from low to high (the most-to-least likely of being flipped)⁴. Then the first rank($\mathbf{H}$) *linearly independent* columns are selected and used to find an error $\hat{\mathbf{e}}$ satisfying the syndrome constraint $\mathbf{H}\hat{\mathbf{e}} = \mathbf{s}$. The unselected values of $\hat{\mathbf{e}}$, associated with the remaining columns of $\mathbf{H}$, are set to zero. This is the zeroth-order OSD (denoted as BP+OSD-0). The combination sweep heuristic [16] for order- λ OSD (denoted as BP+OSD-CS λ) additionally searches over all weight-one configurations of all the unselected VNs, and weight-two patterns of the first λ unselected VNs. BP+OSD may still exhibit an error floor, and the behavior is affected by the number of iterations, the scaling factor, and the scheduling used for the BP preprocessing [24].

B. Circuit-level noise

In [7], along with the parity-check matrices for the bivariate bicycle (BB) codes, a carefully designed gate ordering for the noisy SM circuit is additionally proposed. The SM is repeated for several rounds due to the unreliability of the measurements. A corresponding PCM can be constructed, see Fig. 14 of [25] for a graphical introduction to how such “circuit codes” are created.

Contrary to data qubit noise, where a VN denotes an error on a data qubit and a CN denotes a noiseless syndrome measurement result, for circuit-level noise a VN denotes a *single fault* in the entire (multi-round) SM circuit while a CN denotes a detector, which is the XOR of the measurement results of a given parity-check from two consecutive rounds. The presence of one in column i and row j of the circuit code PCM means the i^{th} fault mechanism triggers the j^{th} detector.

As an example, when repeating the SM circuit for R rounds, consider one syndrome check c whose measurement results from oldest to most recent in time are $m_1, m_2, \dots, m_R$. The

resulting detectors associated with this check c are checking $m_1, m_2 \oplus m_1, m_3 \oplus m_2, \dots, m_R \oplus m_{R-1}$, which will be zero in the absence of any faults. Now consider the VN that denotes a single measurement flip in round $r > 2$ on this particular check c . This VN has degree two because it triggers both detectors $m_{k-1} \oplus m_k$ and $m_{k+1} \oplus m_k$ to one, and it does not trigger detectors associated to other checks. In Fig. 1 showing the circuit code PCM $\mathbf{H}_{\text{circuit}}$, we group rows (detectors) by round. The top block of rows (the rows that the top $\mathbf{H}_0$ occupies) are detectors like m_1 associated to all checks, then the next block of rows are $m_2 \oplus m_1$ like detectors, and so on. The bottom block of rows are the most recent detector values.

Further in Fig. 1, one can see that there are two kinds of faults. One kind triggers detectors in the current round (block of rows) only; these faults form the columns of $\mathbf{H}_0$. An example is a fault on a data qubit immediately prior to a round of syndrome extraction, which causes check values to change only in that round. The other kind of fault triggers detectors in consecutive two rounds, constituting the columns of $\begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{bmatrix}$. Single measurement faults are an example, as well as the various faults in the middle of the SM circuit. The latter are captured by only a subset of the syndrome checks in the occurring round, but can be fully captured by all subsequent rounds. Using the same argument, one can see that no single fault triggers detectors from more than two rounds, thanks to the sparsification created by the XOR operation.

Fault mechanisms can be combined if they trigger the same set of detectors, have the same noiseless syndrome at the end, and induce the same logical error. A simple example is an X-flip on the target qubit before and after a CNOT gate. The columns of such equivalent fault mechanisms are merged, and a new prior probability is calculated for the new VN. In this linearized model [7], the possible correlation between columns is ignored, meaning the different single fault mechanisms are assumed to happen independently. Given all the detector values and the final noiseless syndrome, the decoder aims to find the most probable set of faults that explains the measurement result. Then the correction operator on the data qubits can be determined by adding up the effects on the output data qubits caused by each selected fault.

Several undesired structures in the circuit code Tanner graph harm its BP performance. An X-flip on the control qubit propagates to both control and target qubits after the CNOT. Consider the triplet of columns representing these three single-fault mechanisms. The XOR of any two columns equals the other, which creates numerous short cycles. The aforementioned measurement fault triggers two detectors and corresponds to a VN of degree two, which is known to cause high error floors in BP decoding [26]. To alleviate the syndrome inconsistency problem, BP+OSD is employed in [7] to perform global decoding of the fault mechanisms that happened in d rounds of noisy syndrome measurements, where d is the code distance. This method is not likely to fulfill the latency constraint, due to the large PCM of the circuit code and the high worst-case runtime of OSD.

In this work, we use the sliding window decoder that is

⁴The binary BP+OSD proposed in Section 3.1 of [5] ranks columns according to reliabilities, i.e., the absolute value of posterior LLR, then solve the equation on least reliable columns by first fixing the rest to their BP hard decisions. [16] ranks according to likelihood of being flipped, i.e., the value of posterior LLR, which is the approach we follow here.

suitable for handling measurement data coming in a streaming fashion. In fact, the XOR of check results from consecutive two SM rounds can be seen as naturally creating a spatially-coupled LDPC (SC-LDPC)-like code, and the sliding window decoder is a standard decoding technique for *classical* SC-LDPC codes [26].

III. SLIDING WINDOW DECODING

We demonstrate the sliding-window decoding of the bivariate bicycle (BB) code family implemented using the proposed syndrome measurement (SM) circuit in Fig. 7 of [7]. X-type and Z-type check operators are decoded separately as in [7], though circuit-level depolarizing noise is assumed. Correlations between X and Z errors could be considered; however, we find that in this case fewer fault mechanisms can be combined, and the PCM of the circuit code becomes significantly larger. In particular, we find the resulting PCM has roughly eight times more columns.

We implement a memory experiment in the Z basis in Stim [27], using the repeated SM circuit from Fig. 7 of [7]. We find that this family of codes possesses a general description of their corresponding circuit codes in terms of the shape of $\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2$, see the caption of Fig 1.

In the (3,1)-sliding window decoding of this circuit code, three rounds of detector values are handled in a window and the window moves downward by one round each time. Inside each window, an inner decoder is applied, e.g., BP+OSD-CS10 or GDG. The decisions for the columns not contained in the next window are committed, and the detector values for the next window are updated.

For example, assume detector values $s_1, s_2, s_3, \dots$ are observed, where each s_i is associated to the i^{th} block of rows. We first use the first window (green) PCM $\mathbf{H}_{win}$ to solve the following syndrome equation (and all subsequent ones) over the binary field,

$$\begin{bmatrix} \mathbf{H}_0 & \mathbf{H}_1 & & & \\ & \mathbf{H}_2 & \mathbf{H}_0 & \mathbf{H}_1 & \\ & & \mathbf{H}_2 & \mathbf{H}_0 & \mathbf{H}_1 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{e}}_0 \\ \hat{\mathbf{e}}_1 \\ \vdots \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}. \quad (6)$$

Here $\hat{\mathbf{e}}_0$ and $\hat{\mathbf{e}}_1$ are the estimated VN patterns associated with the columns of leftmost $\mathbf{H}_0$ and $\mathbf{H}_1$ respectively, and they are the columns that we commit to before moving to the next window.

It is clear that if an inner decoder gives a solution that satisfies this equation, then

$$[\mathbf{H}_0 \quad \mathbf{H}_1] \begin{bmatrix} \hat{\mathbf{e}}_0 \\ \hat{\mathbf{e}}_1 \end{bmatrix} = [s_1] \quad (7)$$

is naturally satisfied. BP+OSD can always find such a solution, however, by applying plain BP alone to this window decoding problem, we empirically observed that the convergence is weak for the BB code family. Importantly, if Eq. (7) is not satisfied, then the overall syndrome equation

$$\mathbf{H}_{circ} \begin{bmatrix} \hat{\mathbf{e}}_0 \\ \hat{\mathbf{e}}_1 \\ \vdots \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \end{bmatrix} \quad (8)$$

can never be satisfied, regardless of subsequent window decoding.

By committing to $\hat{\mathbf{e}}_0$ and $\hat{\mathbf{e}}_1$, we need to update s_2 used for the next window by $s'_2 = s_2 + \mathbf{H}_2 \hat{\mathbf{e}}_1$, then use s'_2, s_3, s_4 to decode the next window. If all partial equations like Eq. (7) hold with respect to newly committed columns and (updated) syndrome, then Eq. (8) will also be satisfied, and vice versa. When employing our GDG decoder for window decoding, we always count Eq. (8) failures into logical errors, besides requiring the overall logical errors caused by the chosen VNs to match the one that Stim [27] returns.

More concretely, consider a simulation for protection of the logical Z information in a $[[N, K, d]]$ code that encodes K logical qubits. A logical Z observable matrix [25] $\mathbf{L}$ consisting of K rows and the same number of columns as $\mathbf{H}_{circ}$ can be constructed as follows. Imagine adding K checks associated to the logical operators to the very end of the multi-round SM circuit. For the single-fault represented by column i of $\mathbf{H}_{circ}$, propagate it to the end of the SM circuit, just before the logical checks. Put a one in the i^{th} columns and j^{th} row of $\mathbf{L}$ if this final error string does not commute with (triggers) the logical check j . For numerical simulation, Stim [27] provides K bits $l_1, \dots, l_K$ indicating the values of the final logical checks, along with the detector values mentioned earlier. We decode using the detector values only and determine if our final estimation satisfies

$$\mathbf{L} \begin{bmatrix} \hat{\mathbf{e}}_0 \\ \hat{\mathbf{e}}_1 \\ \vdots \end{bmatrix} = \begin{bmatrix} l_1 \\ \vdots \\ l_K \end{bmatrix} \quad (9)$$

and Eq. (8). The decoding is deemed successful if both equations are satisfied.

In classical SC-LDPC code decoding, BP is used on each window, and usually a small number of iterations of BP already leads to convergence. As mentioned above, this is not the case for $\mathbf{H}_{circ}$ of the BB family. Therefore, we apply BP+OSD-CS10 to each window and benchmark the performance against the global decoding over distance d rounds of SM, see Fig. 3.

Though each window deals with a much smaller PCM compared to global decoding over d rounds, fulfilling the stringent latency constraint is still challenging. Before proceeding to the next section where we introduce the GDG decoder, we make a few comments on the windowed approach.

Each window is extremely wide, i.e., the number of columns is much larger than the number of rows. To improve efficiency, we merge the columns of the bottom right (full-rank) matrix $\mathbf{H}_1$ in each window to an identity matrix and recalculate priors. Secondly, inspired by OSD-0, we notice that certain columns can be dropped (decided to zero) after BP posterior LLR ranking. For a window of w' rows, instead of testing for linear dependency and choosing the first $\text{rank}(\mathbf{H}) \approx w'$ columns as in OSD-0, we *skip the test* and simply choose the first $2w'$ columns for post-processing. This method is always used together with GDG in the next section. As it turns out, the first $2w'$ columns are almost always enough for the syndrome

Algorithm 1 Select a VN to decimate and compute the favored value based on LLR history

Input Depth D
Output Chosen VN index g , favored guess value f

```

1: procedure SELECT-VN( $D$ )
2:    $L_{min} \leftarrow \infty$ 
3:    $g_{min} \leftarrow -1$ 
4:    $\tilde{L}_{min} \leftarrow \infty$  // for all negative LLR history
5:    $\tilde{g}_{min} \leftarrow -1$ 
6:   for all VN  $v_i$  do
7:     if  $vn\_status[i] \neq -1$  then // decimated
8:       continue
9:     if  $\deg(v_i) \leq 2$  then
10:      continue
11:    // history: last 4 iter. of Eq. (2)
12:     $\Lambda_{hist} = [\Lambda_i^{(t-3)}, \Lambda_i^{(t-2)}, \Lambda_i^{(t-1)}, \Lambda_i^{(t)}]$ 
13:     $L_{sum} \leftarrow \text{sum}(\Lambda_{hist})$ 
14:    if not in low error mode then
15:       $b \leftarrow \text{AGG-DEC}(\Lambda_{hist}, D)$  // Alg. 3
16:      if  $b \neq -1$  then //  $\Lambda_{hist}$  reliable
17:         $vn\_status[i] \leftarrow \hat{e}[i] \leftarrow b$ 
18:        continue //  $v_i$  quits selection
19:      if  $L_{sum} < L_{min}$  then
20:         $L_{min} \leftarrow L_{sum}$ 
21:         $g_{min} \leftarrow i$ 
22:      if  $\Lambda_{hist} \leq 0$  and  $L_{sum} < \tilde{L}_{min}$  then
23:         $\tilde{L}_{min} \leftarrow L_{sum}$ 
24:         $\tilde{g}_{min} \leftarrow i$ 
25:    // select one VN and favored value for it to guess
26:    if  $\tilde{g}_{min} \neq -1$  then
27:       $g \leftarrow \tilde{g}_{min}$ 
28:       $f \leftarrow 1$ 
29:    else
30:       $g \leftarrow g_{min}$ 
31:       $f \leftarrow \text{sign}(L_{min})$ 
32:  return  $g, f$ 

```

(blue squiggles in Figure 2), which are decision paths that directly split off from the main branch. These decimate a VN contrary to the sign of the LLR history only at *one point*, where the branch splits, and all subsequent decisions follow the sign. These side branches improve convergence in ensemble decoding significantly, and by incorporating them into the inner window decoder, we achieve BP+OSD-0 performance. Finally, inspired by the pattern exploration of unchosen VNs in high-order OSD, we add the *tree branches* (green squiggles in Figure 2) emanating from the leaves of a depth-4 *guessing tree*⁵. They explore all the decimation values of the first four decisions. After splitting, they all proceed following the sign. The tree branches help in finding a more probable (smaller path metric) estimated error pattern and they bridge us to

⁵Following line 22-24 of Alg. 2, the id labeled for tree branches in Fig. 2 is only a demonstration for a guessing tree where the favored value at each step is one, which is not always the case in reality.

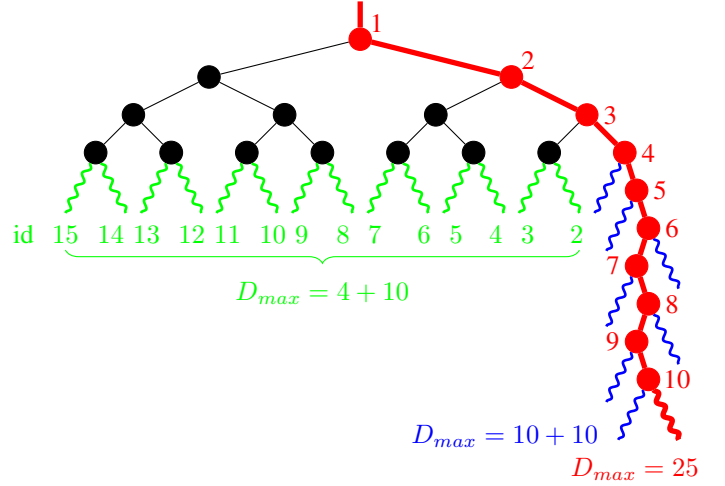


Fig. 2. The decision tree for the BPGDG algorithm. VN selection and decimation are made after each step, where one step is defined to be six BP iterations. The red path is the main branch. The solid dots (red or black) are the only places where guessing is allowed. No splitting from the main branch into side branches (blue) after reaching depth 10. No guessing for the tree branches (green) after depth 4. The main branch terminates after 25 steps ($D_{max} = 25$), regardless of convergence. The side branches are allowed to run 10 more steps after their split-off at depth D_{splitt} , i.e., $D_{max} = D_{splitt} + 10$. The tree branches are also allowed to run 10 more steps after splitting from their neighbors at depth 4, which means $D_{max} = 4 + 10$ for all tree branches.

BP+OSD-CS10 performance. We name the resulting decoder a guided decimation guessing (GDG) decoder.

Let us summarize the differences between GDG and BPGD [12] decoder. First, we change the VN selection rule and make decisions based on LLR history. Second, we allow guessing on the main decision branch and at very early steps. The guessing tree has a similar spirit to guessing decoding on classical erasure channel [32], [33] and later AWGN channel [34]. However, to our knowledge, no prior work has proposed the side branches, which is the key to convergence for our VN selection rule. It is probably because prior works decimate the most reliable VNs and guessing is unnecessary.

C. Simulation

In Fig. 3, we show the performance of using the (3,1) sliding window to decode the BB codes on circuit-level noise. All BP runs involved use min-sum update, flooding scheduling, and scaling factor $\alpha = 1.0$. As described at the end of the previous section, for each window we first run BP on the original PCM for eight or sixteen iterations if $N \leq 144$ or $N = 288$, rank columns according to the sum of LLR posteriors from the latest four iterations, and apply GDG to the first $2w'$ columns.

For $N \leq 144$, the decision tree with parameters shown in Fig. 2 is used as the inner decoder. The GDG critical path is $150 = 25 \cdot 6$ BP iterations. The worst-case runtime of each window is measured to be $\sim 3ms$ for $N = 144$ with a multi-thread implementation on an Intel i9-13900K CPU, while 200 BP iterations + OSD-CS10 on the original PCM takes $\sim 20ms$ or $\sim 10ms$ on the shortened PCM in

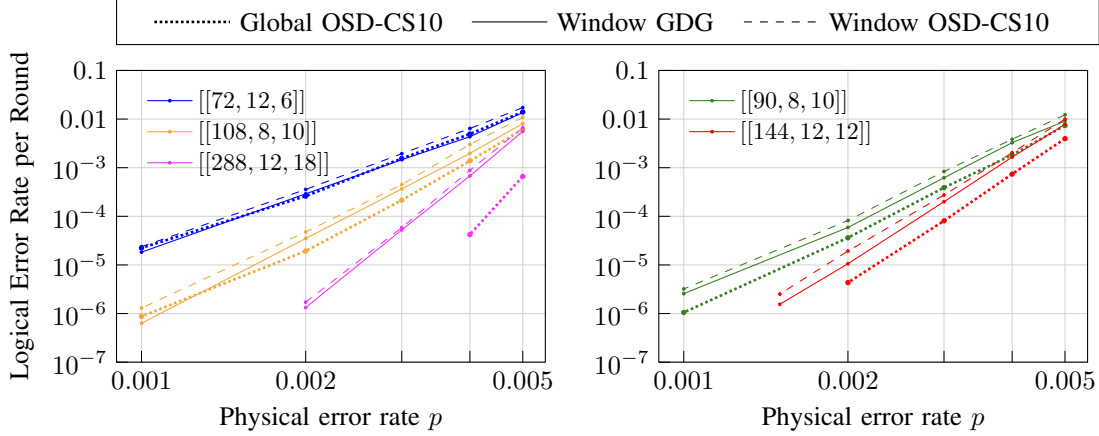


Fig. 3. Logical error rate per round. The syndrome measurement is repeated d rounds for a distance d code. Dotted lines are global decoding over d rounds using BP (1000 iterations) + OSD-CS10. Dashed lines are (3,1)-sliding window decoding where each window uses BP (200 iterations) + OSD-CS10 as the inner decoder. Solid lines use GDG as the inner decoder in (3,1)-sliding window. For GDG, low error mode (no aggressive decimation) is used for $p \leq 0.002$ for all codes.

Algorithm 2 Belief Propagation Guided Decimation Guessing

Input Parity-check matrix $\mathbf{H}$, syndrome $\mathbf{s}$
Output Estimated error $\hat{\mathbf{e}}$

```

1: for all decision paths in Fig. 2 do
2:   for all VN  $v_i$  do // initialization status mask
3:      $vn\_status[i] \leftarrow -1$  // not yet decimated
4:   for  $D \leftarrow 1, \dots, D_{max}$  do
5:      $\hat{\mathbf{e}} \leftarrow$  run BP for 6 iterations
6:     if  $\mathbf{H}\hat{\mathbf{e}} = \mathbf{s}$  then
7:       calculate path metric for  $\hat{\mathbf{e}}$  with Eq. (5)
8:       break
9:     // get VN index  $g$  and favored guess value  $f$ 
10:     $g, f \leftarrow \text{SELECT-VN}(D)$  // Alg. 1
11:    if is main branch then
12:       $vn\_status[g] \leftarrow \hat{\mathbf{e}}[g] \leftarrow f$ 
13:    else if is side branch then
14:      if  $D \neq D_{splitt}$  then
15:         $vn\_status[g] \leftarrow \hat{\mathbf{e}}[g] \leftarrow f$ 
16:      else
17:         $vn\_status[g] \leftarrow \hat{\mathbf{e}}[g] \leftarrow 1 - f$ 
18:    else // tree branch
19:      if  $D > 4$  then
20:         $vn\_status[g] \leftarrow \hat{\mathbf{e}}[g] \leftarrow f$ 
21:      else
22:        // decimate accord. to the binary repr. of id
23:         $b \leftarrow$  the  $D^{th}$  bit of  $\text{id} \in [2, 15]$ 
24:         $vn\_status[g] \leftarrow \hat{\mathbf{e}}[g] \leftarrow b + (-1)^b f$ 
25: return the  $\hat{\mathbf{e}}$  with the smallest path metric

```

the worst-case. Note that runtime on CPU is not indicative of performance on specialized hardware. Since we use BP with flooding scheduling, GDG is expected to benefit from the inherent parallel CN/VN updates on FPGA or GPU, though the VN selections (*argmin* operations) may become

the bottleneck. Similarly, the $\text{rank}(\mathbf{H})$ pivot findings⁶ in OSD are the bottleneck operations on ASICs. In this sense, GDG may still have a runtime advantage when parallelization is considered, since the number of required *argmin* operations decreases with physical error rates. For the $N = 288$ code, a maximum depth of 40 is used for the main branch, and the side/tree branches are allowed to run for 20 more steps after splitting at depth 20 or 5. With the same decision tree parameters applied to (5, 2)-sliding window GDG decoding for $N = 144$, we get a slightly better logical error rate than the global decoding in Fig. 3, indicating order 10 is not sufficient enough for global OSD-CS decoding. Our global OSD-CS decoding curves match those in Fig. 3 from [7] quite well, except for the $N = 288$ code. The performance degradation is possibly also due to our insufficient OSD-CS order. We also tried (4, 1)-window GDG for $N = 288$, it is significantly closer to the global decoding curve in Fig. 3 than the one from (3, 1) decoding. The specific tree parameters and performance are available online.

D. Do not waste guesses

After each step of decimation, the remaining Tanner graph can have degree one CNs. The single VN connected to such a CN can be uniquely decoded as the current syndrome of this CN. This *peeling* step is repeated until no degree one CN is present. Peeling adds a small overhead to the various decision paths, but it crucially prevents GDG from wasting decision steps on these apparently decidable VNs.

As shown in Fig. 3, when used as the inner decoder of a (3,1)-sliding window decoder, the GDG decoder gives favorable performance for physical error rates $p \leq 0.002$. For slightly higher physical error rates, there is a higher chance that the number of single-fault mechanisms that occurred is larger than the maximum depth of the GDG. To improve

⁶This argument is for higher-order OSD. For OSD-0, there is no need to do full Gaussian elimination, adding in columns can stop once the syndrome is in the span of the selected ones, see Alg. 2 of [5].

Algorithm 3 Aggressive Decimation

Input LLR history Λ_{hist} of one VN, depth D

Output Decimation value for this VN

```
1: procedure AGG-DEC( $\Lambda_{hist}$ ,  $D$ )
2:    $P_A \leftarrow -3$  if on main branch else 0
3:    $P_B \leftarrow -12$  if on main branch else  $-10$ 
4:    $P_B \leftarrow -16$  if  $D = 1$ 
5:    $P_C \leftarrow 30$ 
6:    $P_D \leftarrow 3$ 
7:   if  $\Lambda_{hist} < P_A$  and  $\text{sum}(\Lambda_{hist}) < P_B$  then
8:     return 1
9:   if ( $\Lambda_{hist} > P_C$  and  $D \leq 4$ ) or
10:  ( $\Lambda_{hist} > P_D$  and  $\exists \geq 3$  unsatisfied CN neigh.) then
11:    return 0
12:  // otherwise  $\Lambda_{hist}$  not reliable enough for decision
13:  return -1
```

convergence with the same maximum depth, we leave the low error mode and use Alg. 3 aggressive decimation in Alg. 1. For each VN, if its LLR history is very positive or very negative, it is simply decimated to zero or one, without performing any guessing. Line 10 of Alg. 3 is inspired by [34], which says VNs that are connected to many unsatisfied CN neighbors are places that lack information. Some effort was spent on finding P_A, P_B, P_C, P_D at $p = 0.003$ for the $N = 144$ code by first looking at posterior LLR distribution in value and then fine-tuning. The same parameters are later applied to all other codes and p , without any further tuning. It is also important to mention that we always clip the VN to CN messages to $[-50, 50]$. The clipping values especially affect P_C .

We also tried Alg. 3 in hindsight for $p \leq 0.002$. However, it causes slight error floors because some VNs are decimated to zero too early. Furthermore, the BP preprocessing iteration on the original window PCM is kept low (8 or 16), otherwise error floors appear. The intuition is that GDG prefers negativity in the messages, and longer BP runs that do not eventually converge ruin the negativity in the message-passing network, especially at low p , where most messages are positive.

V. CONCLUSION

We introduce the GDG decoder in this work and use it as the inner decoder of a sliding window decoder for BB codes under circuit-level noise. In particular, we limit the critical path of this decoder to 150 BP iterations on each window for (3,1)-decoding of the $N = 144$ code, and nevertheless GDG achieves excellent logical error rates. However, GDG is not a general-purpose decoder like OSD. Rather, it is designed for the sub-threshold region. The average number of error weights should be smaller than the maximum step for GDG to be utilized to its full potential. For QLDPC codes, the error floor [35] is a conundrum, and GDG awaits special hardware implementation to investigate its error floor behavior.

Furthermore, we design GDG for syndrome decoding, not codeword decoding, e.g. as in Steane error correction [36]. We believe a similar VN selection rule for codeword decoding can

be developed based on the history of $\text{sign}(\Lambda_i) \cdot \Lambda_i^{(t)}$. Lastly, though we draw an analogy to classical SC-LDPC window decoding, we did not exploit their key feature of reusing BP messages from the previous window in the overlapping region to initialize the next window. This would be possible for the GDG decoder, using the BP internal messages from the branch with the minimum path metric, whereas no soft information can be retained for the next window when using BP+OSD. We leave such investigations for future work.

ACKNOWLEDGMENT

We thank Pavel Pantelev for the idea of adding a little bit of globalness to a local decoder like BP, leading us to explore BPGD in the first place. The global operation turns out to be selecting the most likely to flip variable node for us. We also thank Joshua Vizslai for useful discussions.

REFERENCES

- [1] P. Pantelev and G. Kalachev, “Quantum LDPC codes with almost linear minimum distance,” *IEEE Transactions on Information Theory*, vol. 68, no. 1, pp. 213–229, 2021.
- [2] —, “Asymptotically good quantum and locally testable classical LDPC codes,” in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, 2022, p. 375–388.
- [3] N. P. Breuckmann and J. N. Eberhardt, “Balanced product quantum codes,” *IEEE Transactions on Information Theory*, vol. 67, no. 10, pp. 6653–6674, 2021.
- [4] A. Leverrier and G. Zémor, “Quantum Tanner codes,” in *IEEE 63rd Annual Symposium on Foundations of Computer Science*, 2022, pp. 872–883.
- [5] P. Pantelev and G. Kalachev, “Degenerate Quantum LDPC Codes With Good Finite Length Performance,” *Quantum*, vol. 5, p. 585, 2021.
- [6] H.-K. Lin and L. P. Pryadko, “Quantum two-block group algebra codes,” *Phys. Rev. A*, vol. 109, p. 022407, 2024.
- [7] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, “High-threshold and low-overhead fault-tolerant quantum memory,” *arXiv preprint arXiv:2308.07915*, 2023.
- [8] Q. Xu, J. P. B. Ataiades, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasic, M. D. Lukin, L. Jiang, and H. Zhou, “Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays,” *arXiv preprint arXiv:2308.08648*, 2023.
- [9] J. Vizslai, W. Yang, S. F. Lin, J. Liu, N. Nottingham, J. M. Baker, and F. T. Chong, “Matching generalized-bicycle codes to neutral atoms for low-overhead fault-tolerance,” *arXiv preprint arXiv:2311.16980*, 2024.
- [10] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataiades, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, “Logical quantum processor based on reconfigurable atom arrays,” *Nature*, vol. 626, no. 7997, p. 58–65, 2023.
- [11] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, “Topological quantum memory,” *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.
- [12] H. Yao, W. A. Laban, C. Häger, A. G. i Amat, and H. D. Pfister, “Belief propagation decoding of quantum LDPC codes with guided decimation,” *arXiv preprint arXiv:2312.10950*, 2023.
- [13] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen, “Scalable surface-code decoders with parallelization in time,” *PRX Quantum*, vol. 4, p. 040344, 2023.
- [14] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell, “Parallel window decoding enables scalable fault tolerant quantum computation,” *Nature Communications*, vol. 14, p. 7040, 2023.
- [15] S. Huang and S. Puri, “Improved noisy syndrome decoding of quantum ldpc codes with sliding window,” *arXiv preprint arXiv:2311.03307*, 2023.
- [16] J. Roffe, D. R. White, S. Burton, and E. Campbell, “Decoding across the quantum low-density parity-check code landscape,” *Phys. Rev. Res.*, vol. 2, p. 043423, 2020.

- [17] A. G. Fowler, A. M. Stephens, and P. Groszkowski, "High-threshold universal quantum computation on the surface code," *Phys. Rev. A*, vol. 80, p. 052312, 2009.
- [18] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Physical Review A*, vol. 54, no. 2, p. 1098, 1996.
- [19] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society A*, vol. 452, no. 1954, pp. 2551–2577, 1996.
- [20] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.
- [21] Z. Babar, P. Botsinis, D. Alanis, S. X. Ng, and L. Hanzo, "Fifteen years of quantum LDPC coding and improved decoding strategies," *IEEE Access*, vol. 3, pp. 2492–2519, 2015.
- [22] D. Poulin and Y. Chung, "On the iterative decoding of sparse quantum codes," *Quantum Information and Computation*, vol. 8, no. 10, pp. 987–1000, 2008.
- [23] N. Raveendran and B. Vasić, "Trapping sets of quantum LDPC codes," *Quantum*, vol. 5, p. 562, 2021.
- [24] J. Du Crest, M. Mhalla, and V. Savin, "Stabilizer inactivation for message-passing decoding of quantum LDPC codes," in *IEEE Information Theory Workshop*, 2022, pp. 488–493.
- [25] O. Higgott and C. Gidney, "Sparse blossom: correcting a million errors per core second with minimum-weight matching," *arXiv preprint arXiv:2303.15933*, 2023.
- [26] A. R. Iyengar, P. H. Siegel, R. L. Urbanke, and J. K. Wolf, "Windowed decoding of spatially coupled codes," *IEEE Transactions on Information Theory*, vol. 59, no. 4, pp. 2277–2292, 2013.
- [27] C. Gidney, "Stim: a fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, 2021.
- [28] M. Mezard, G. Parisi, and R. Zecchina, "Analytic and algorithmic solution of random satisfiability problems," *Science*, vol. 297, pp. 812–815, 2002.
- [29] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian, "Solving constraint satisfaction problems through belief propagation-guided decimation," *45th Annual Allerton Conference on Communication, Control, and Computing*, vol. 1, 2007.
- [30] V. Aref, N. Macris, and M. Vuffray, "Approaching the rate-distortion limit with spatial coupling, belief propagation, and decimation," *IEEE Transactions on Information Theory*, vol. 61, no. 7, pp. 3954–3979, 2015.
- [31] S. Cammerer, M. Ebada, A. Elkelesh, and S. ten Brink, "Sparse graphs for belief propagation decoding of polar codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1465–1469.
- [32] H. Pishro-Nik and F. Fekri, "On decoding of low-density parity-check codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 3, pp. 439–454, 2004.
- [33] C. Measson, A. Montanari, and R. Urbanke, "Maxwell construction: The hidden bridge between iterative and maximum a posteriori decoding," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5277–5307, 2008.
- [34] H. Pishro-Nik and F. Fekri, "Results on punctured low-density parity-check codes and improved iterative decoding techniques," *IEEE Transactions on Information Theory*, vol. 53, no. 2, pp. 599–614, 2007.
- [35] T. Richardson, "Error floors of LDPC codes," in *Proceedings of the annual Allerton conference on Communication Control and Computing*, 2003.
- [36] A. M. Steane, "Active stabilization, quantum computation, and quantum state synthesis," *Phys. Rev. Lett.*, vol. 78, pp. 2252–2255, 1997.
- [37] N. Raveendran, N. Rengaswamy, A. Pradhan, and B. Vasić, "Soft syndrome decoding of quantum LDPC codes for joint correction of data and syndrome errors," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2022, pp. 275–281.
- [38] J. Roffe, "LDPC: Python tools for low density parity check codes," 2022. [Online]. Available: <https://github.com/quantumgizmos/ldpc>

APPENDIX A DATA QUBIT NOISE

In Fig. 4, we show the logical error rate of employing GDG or BP+OSD on data qubit noise. Here we focus on the X type of noise only and assume each data qubit is subject to i.i.d. bit flip with probability p_d . This equals the classical binary

symmetric channel $BSC(p_d)$. The scaling factor used for GDG is always set to 0.625. For OSD methods, the best scaling factor is chosen among $\{0.5, 0.625, 0.8, 1.0\}$. The H_X and H_Z have girth 6 for these BB codes, when decoding X and Z separately, though we still use a history length of 4 for GDG as in the main text. The three methods are run over the same set of generated noise, and for the $N = 288$ at $p_d = 0.02$, OSD-CS10 shows an error floor while GDG does not.

We did not perform simulation on data depolarizing channels, where each data qubit independently and identically experiences a random X , Z , or $Y = iXZ$ type of error, each with probability $p/3$ for some $p \in [0, 1]^7$, as a quaternary BP (BP4) version for GDG should be designed and compared to BP4+OSD⁸.

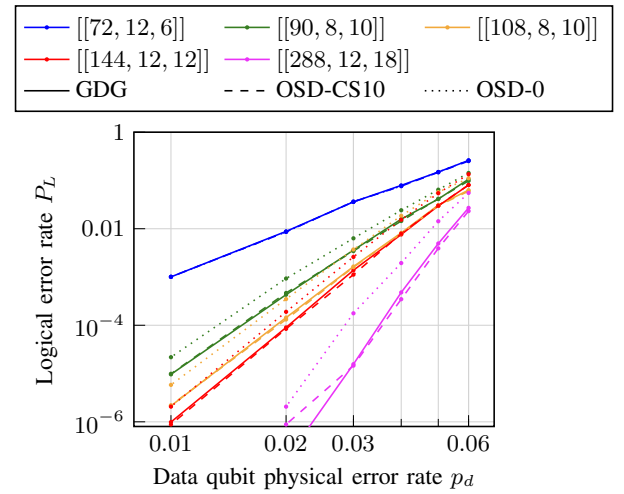


Fig. 4. Data qubit noise, X-noise only. Solid lines are GDG in low error mode with main branch maximum depth 40, side or tree branches do not split after depth 20 or 5 and are allowed to proceed for 30 more steps. Scaling factor 0.625. Dashed and dotted lines are BP+OSD-CS10 and BP+OSD-0 respectively. BP preprocessing for both OSD methods is 100 iterations. Scaling factor 0.5 is used for $N \leq 144$ and 0.625 is used for $N = 288$.

APPENDIX B SINGLE SHOT SYNDROME NOISE

In the main text, so far we have been investigating using BB codes as a fault-tolerant memory. The simulation we performed is also an emulation of the memory experiment. The final round of the noiseless syndrome measurement can be obtained by measuring (collapsing) all the data qubits at the end and followed by *classical* syndrome calculation. However, when an error correction code is used in a real-time experiment, it is not clear how to obtain noiseless syndrome, since the data qubits shall not be collapsed before a non-Clifford gate and we have to deal with measurement noise.

The lift-product construction [1] creates a naturally over-complete PCM. The GHP codes in [5], 2BGA codes [6] and

⁷As a rule of thumb, decoding on $Depolarize(p)$ is easier to decode than $BSC(2p/3)$ since the correlation between X and Z can be exploited.

⁸An implementation can be found in our repository online. Additionally, our TensorFlow implementation of BP4+OSD-0 can be found in <https://github.com/gongaa/Feedback-GNN/blob/main/examples/OSD.ipynb>.

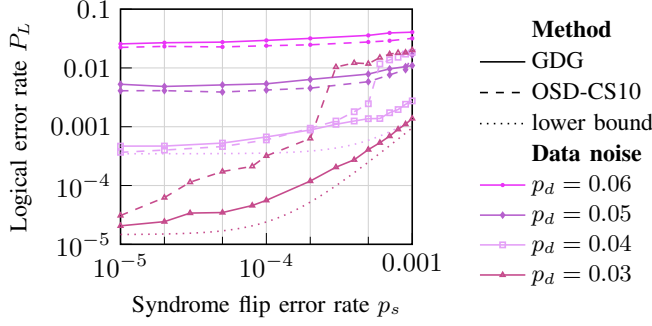


Fig. 5. Logical error rate for the $[[288, 12, 18]]$ code at data qubit X-noise $p_d = 0.03 \sim 0.06$ and i.i.d. syndrome bit-flip with probability $p_s \in [10^{-5}, 10^{-3}]$. The dotted lines are the lower bound $p_L(p_d) + 864 \cdot p_s^2 + 2592 \cdot p_d \cdot p_s^2$.

the BB codes [7] are small special cases. In this section, we investigate the syndrome protection offered by the syndrome code in the single-shot noisy syndrome measurement setting for the $[[288, 12, 18]]$ BB code.

In Fig. 5, assume the data qubit noise is subject to $\mathbf{e} \sim \text{BSC}(p_d)$ and the syndrome noise is subject to $\mathbf{s}' \sim \text{BSC}(p_s)$. Throughout we assume $p_s < p_d$, otherwise it makes no sense to perform measurements. The observed syndrome is thus $\mathbf{H}_X \mathbf{e} + \mathbf{s}'$ and the decoding $\hat{\mathbf{e}}$ is deemed successful if $\mathbf{H}_Z^\perp(\hat{\mathbf{e}} + \mathbf{e}) = \mathbf{0}$. The row span of $\mathbf{H}_Z^\perp$ contains $\mathbf{H}_X$ by the CSS constraint, and thus $\hat{\mathbf{e}}$ needs to have the same syndrome as $\mathbf{e}$ for the decoding to be correct. In Tanner graph construction, we add one virtual VN to each CN, the flipping of such a virtual VN implies a syndrome flip on the associated CN. The effective PCM is then $[\mathbf{H}_X | \mathbf{I}]$. The prior LLR to the virtual and original VNs are $\log \frac{1-p_s}{p_s}$ and $\log \frac{1-p_d}{p_d}$, respectively. For the $[[288, 12, 18]]$ code with the PCMs specified in [7], there are 288 original VNs of degree three and 144 virtual VNs of degree one. We employ BP+OSD-CS10 and GDG to decode $\hat{\mathbf{e}}$. The same parameters as in Fig. 4 are used for both decoders.

Next, we discuss how to obtain the lower bound for the logical error rate in Fig. 5. The first term $p_L(p_d)$ is the logical error rate in the absence of syndrome error⁹, which we obtain from Fig. 4. The second term $864 \cdot p_s^2 = \binom{3}{2} \cdot 288 \cdot p_s^2$ comes from the configurations in Fig. 6(a) where weight-two syndrome error happens on the three CNs neighbors of an original VN. The third term $2592 \cdot p_d \cdot p_s^2$ originates from a weight-one data qubit error and weight-two syndrome error in configuration Fig. 6(b). This term requires careful counting for the coefficient.

Consider the $\mathbf{H}_X = [\mathbf{A} | \mathbf{B}]$ matrix of the $N = 288$ BB code, where $\mathbf{A} = x^3 + y^2 + y^7$, $\mathbf{B} = y^3 + x + x^2$ and $x = \mathbf{S}_{12} \otimes \mathbf{I}_{12}$, $y = \mathbf{I}_{12} \otimes \mathbf{S}_{12}$, $\mathbf{S}_{12}$ is the cyclic permuting matrix. The syndrome code for $\mathbf{H}_X$ is its column span. Its shape 144×288 implies that each column is a codeword of length 144 and the syndrome code is spanned by its 288 linearly dependent columns, each of weight-three. The

⁹Here we ignore wrong decoding with a wrong denoised syndrome but accidentally lead to the correct answer.

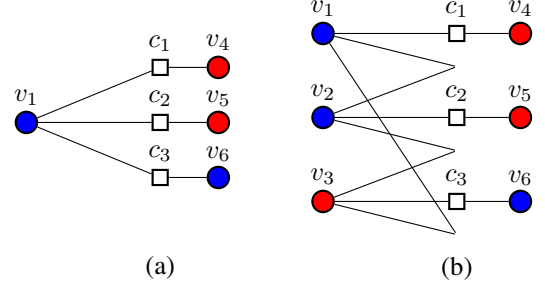


Fig. 6. Tanner graph of the data and syndrome noise decoding, only showing relevant VNs/CNs. The VNs on the left of CNs are data qubits, and the VNs on the right of CNs are virtual nodes for syndrome noise. The red and blue configurations cause the same syndrome. If the red configuration happens, a maximum likelihood decoder will choose the more probable blue one because $p_d > p_s$ and get the denoised syndrome wrong, introducing logical errors.

syndrome code can be written as the ideal (b_1, b_2) in the quotient ring $R = \mathbb{F}_2[x, y]/(x^{12} = 1, y^{12} = 1)$. The two base polynomials $b_1 = y^5 + y^{10} + x^9$ and $b_2 = y^9 + x^{10} + x^{11}$ are the first columns of matrix $\mathbf{A}$ and $\mathbf{B}$ respectively. A monomial $x^a y^b$ present in the polynomial indicates a one at position $12a + b$. Then the 288 columns of $\mathbf{H}_X$ are just $b_1 \cdot x^a y^b$ and $b_2 \cdot x^a y^b$, $0 \leq a, b \leq 11$, and it is easy to see they all have weight three. Since the syndrome code is a linear code, the distance is upper bound by three. Indeed, the column span of $\mathbf{A}$ and $\mathbf{B}$ each has distance three, while the column span of $\mathbf{H}_X = [\mathbf{A} | \mathbf{B}]$ has distance two. Importantly, this does not imply weight one syndrome errors will cause decoding failure. In fact, to denoise a weight-one syndrome noise to a weight-two syndrome codeword, an additional 14 VNs need to be flipped as well. For the number of configurations in Fig. 6(b), we need to count the number of weight-three syndrome codewords caused by three original VNs. They are $b_1^2 = y^{10} + y^{20} + x^{18} = y^5 \cdot b_1 + y^{10} \cdot b_1 + x^9 \cdot b_1$ or b_2^2 and their 144 shifts by $x^a y^b$. Therefore, all together there are $\binom{3}{2} \cdot \binom{3}{1} \cdot 2 \cdot 144 = 2592$ such configurations.

The above arguments hint that a higher level of syndrome protection may be achieved by a larger distance of the syndrome code, which is upper bound by the column weight of the PCM. The $[[254, 28, 14 \leq d \leq 20]]$ code in [5] is also a bicycle code and the generator polynomials for $\mathbf{A}$ and $\mathbf{B}$ are $a(x) = 1 + x^{15} + x^{20} + x^{28} + x^{66}$ and $b(x) = 1 + x^{58} + x^{59} + x^{100} + x^{121}$, where x is the cyclic permuting matrix of size 127. Its syndrome code has distance five since $g(x) \mid \gcd(a(x), b(x))$ and $g(x) = g_1(x) \cdot g_2(x) = (x^7 + x + 1) \cdot (x^7 + x^5 + x^3 + x + 1)$ is the generator polynomial of the $[[127, 14, 5]]$ BCH code¹⁰. The first dominant term to the logical error rate can be obtained similarly to Fig. 6(a) and is $254 \cdot \binom{5}{3} \cdot p_s^3$. Simulation with GDG shows that this code can achieve $\sim 10^{-7}$ logical error rate at $p_d = 0.01$ and $p_s = 10^{-4}$. On the contrary, the logical error rate for the $[[288, 12, 18]]$ code at $p_s = 10^{-4}$ is at least $864 \cdot p_s^2 \approx 10^{-5}$.

It should be mentioned that GDG can handle soft syndrome

¹⁰The distance lower bound follows from the BCH bound. Assume $g_1(\alpha) = 0$, then α^2, α^4 are also roots. Further, since $g_1(\alpha) \mid g_2(\alpha^3)$, α^3 is also a root of $g(\alpha)$.

information as well, instead of using $\log \frac{1-p_s}{p_s}$ as the prior LLR for the virtual VNs in BSC(p_s), an input-dependent LLR can be used, see Eq. (1) of [37] for Gaussian syndrome noise. Moreover, in Fig. 4(a) of [37], a phase transition was observed when sweeping the standard deviation of the Gaussian noise, despite using a different decoding method on a different code.

APPENDIX C GDG IMPLEMENTATION

We use the `mod2sparse` library for sparse matrix manipulation, the same library was used by the BP+OSD package [38] by Roffe et al. To handle decimations, we additionally maintain a status snapshot containing ternary masks for VNs (decided with 0/1, undecided) and CNs (active with 0/1, resolved). In message passing, CN/VN updates simply ignore messages coming from inactive neighboring VNs/CNs.

To facilitate intermediate peeling steps, we also maintain a vector for CN degrees, which is updated whenever VN decimation is performed and hence is a part of the status snapshot. Peeling stops when all CN degrees are larger than one. It is possible that *contradictions* happen during peeling, e.g., two degree-one CNs want to set contradicting values for their common VN neighbor. In this case, the current decision path is immediately killed and it returns PM ∞ .

There is also a vector for VN degrees so that degree one or two VNs can be easily skipped in Alg. 1. This vector never needs to be updated, since decimation, no matter induced by VN selection or peeling, does not cause the degree of *active* VN to change.

In Fig. 2, since previous decisions affect the later, *different decision paths at the same depth may not decimate the same VN*. Moreover, BP iterations are reused as much as possible. For example, the two neighboring tree branches share the first few decisions, and each side branch has some overlap with the main branch. In this case, at the splitting step, the status snapshot is saved and the selected VN is decimated to the favored value first. After this direction is finished, the status snapshot is reloaded and decimation in the other direction is performed. As only the CN/VN status is saved but not the BP messages for the unfavored direction, the message-passing network has to be re-initialized.

We target an Intel i9-13900K CPU that has 32 logical cores during development. For the multi-thread implementation, apart from the main thread associated to the main branch, we have tree threads that each handle two neighboring tree branches, and side branches each handling a side branch. Each thread is set affinity to one CPU logical core. This assignment enables us to use 32 threads for the $N = 288$ code. Tree threads run independently from the main thread after copying the column-permuted PCM, starting from the root of the decision tree. Side threads copy columns and wait until the main thread writes to it the status snapshot. It is also possible to decouple the side threads from the main thread. They can run from the root and compute the overlapping decision paths themselves due to the deterministic nature of guided decimation. By this means, inter-core communication can also

be minimized. However, we did not follow this approach due to the P-core and E-core differences in this Intel CPU. Side threads are assigned to the slower E-cores for the $N = 288$ codes.

The single-thread version is more straightforward. It first proceeds with the main branch and saves the snapshots along the way. After completion of the main branch, it reloads each snapshot and runs BP, again saving the snapshot if guessing is still allowed. It terminates when all the snapshots are consumed. Since the BP messages are not retained by reloading snapshots, the single-thread version re-initializes BP more often in the guessing tree region. Moreover, some early-stop heuristics are implemented for the single-thread version, improving amortized runtime but causing further differences to the multi-thread version.

APPENDIX D STIM CIRCUIT IMPLEMENTATION

Our Stim [27] circuit implementation nearly follows the surface code memory experiments. When doing the memory experiment in the Z or X basis, all the data qubits are initialized in $|0\rangle$ or $|+\rangle$ state. If R rounds are specified by the user, including the noisy encoding round, the noisy SM circuit is repeated R times. Finally, a noiseless round of SM is added to capture the final error. We implement this step by measuring all the data qubits in the Z or X basis and doing classical syndrome calculations afterward. Different from the surface code implementation, we do not add flips before this final data qubit measurement. Another difference is that we always use RX or MRX , instead of applying noisy Hadamard gates before and after R or MR gates, as it is the practice in [17]. The above differences in circuit creation do not cause changes in the parity-check matrix of the circuit code when X and Z are decoded separately. They only affect the prior probabilities, and GDG is robust to these changes. The logical error rate per round p_L is calculated from the total logical error rate $P_{L,R}$ as $p_L = 1 - (1 - P_{L,R})^{1/R}$.

It is important to mention that we always require both Eq. (8) and Eq. (9) to hold for the decoding to be successful. This stricter requirement may lead to a slight overestimation of the logical error rate for GDG due to potential syndrome inconsistency in Eq. (8). OSD remains unaffected because it has no consistency issue thanks to Gaussian elimination. To see why overestimation can happen for GDG, assume GDG fails Eq. (8) because its estimated fault pattern and the actual one differ *only* by some measurement faults. Measurement faults have no end effect, that is to say, the estimated fault propagates to the same final error string as the actual fault, which of course introduces no logical error. Note that in this case, the estimated fault should satisfy Eq. (9).

However, we believe the overestimation of GDG logical error is negligible, because it is very rare (less than one percent of the total failure cases) that GDG gives an estimation that fails Eq. (8) but satisfies Eq. (9). Even when it happens, OSD-CS10 usually induces a logical error when decoding them,

therefore we tend to believe these rare cases are inherently not decodable.

A more formal definition of the two logical error criteria and the relation between them is as follows. Consider a memory experiment in the Z basis. Call the actual final X-type error string $\mathbf{x}$, which is captured by the noiseless Z-type checks $\mathbf{H}_Z$ added to the end. The estimated final X-type error $\hat{\mathbf{x}}$ does not introduce a logical error if and only if $\hat{\mathbf{x}}$ and $\mathbf{x}$ differ by an X-type stabilizer. This criterion can be written as $\mathbf{H}_X^\perp(\mathbf{x} + \hat{\mathbf{x}}) = \mathbf{0}$. The row span of $\mathbf{H}_X^\perp$ contains $\mathbf{H}_Z$, and $\mathbf{H}_X^\perp \setminus \mathbf{H}_Z$ is the logical Z operator $\mathbf{L}_Z$. Therefore, both $\mathbf{H}_Z(\mathbf{x} + \hat{\mathbf{x}}) = \mathbf{0}$ and $\mathbf{L}_Z(\mathbf{x} + \hat{\mathbf{x}}) = \mathbf{0}$ need to be satisfied. The second equation is just a rephrased version of Eq. (9), but Eq. (8) strictly implies the first equation.

Another subtle thing is that, in Fig. 1, the top left $\mathbf{H}_0$ is actually $\mathbf{H}'_0$ due to the encoding round difference. The columns of $\mathbf{H}'_0$ are contained in $\mathbf{H}_0$ and $\mathbf{H}'_0$ has shape $w \times 2w$. The $2w$ columns come from fault equivalently happening individually on L- or R-data before all CNOT gates in Fig. 7 of [7]. In later SM round, $\mathbf{H}_0$ has w more columns than $\mathbf{H}'_0$, they originate from the fault happening just before the block CNOT gates in round 6 on X-checks in Fig. 7 of [7]. One can see this kind of fault propagates to two faults on L-data qubits before the next cycle starts, and cannot be caught by Z-check measurements in the current cycle.

APPENDIX E MISCELLANEOUS DISCUSSIONS

For the $[[288, 12, 18]]$ code, a noticeable performance degradation between our global decoding curve in Fig. 3 and Fig. 3 from [7] is observed. This is possibly due to the insufficient OSD-CS order used by us. Furthermore, in Fig. 3, the (3,1)-sliding window decoder also has a larger factor of performance loss compared to shorter block-length codes, since the window size 3 is too small compared to the code distance 18. In Fig. 7, one can see that a (4,1)-sliding window decoder manages to bridge the performance gap by half in logarithmic scales. For this high physical error rate 0.005, the GDG on the last

window is replaced by BP+OSD-CS10 in both sliding window decoders, which gives roughly a factor of two performance improvement when SM is repeated for 3 rounds. However, the improvement gradually decreases to almost no effect for 18 rounds.

Throughout our simulation, we find that GDG outperforms BP+OSD-CS10 on all the previous windows, in terms of path metric, but cannot guarantee absolute convergence in the last window where noiseless SM is present, leading to slight performance loss. Replacing GDG with BP+OSD-CS10 in the last window usually gives a little performance boost (less than 20 percent) for the $N \leq 144$ codes, but this boost gets smaller when more SM rounds are used. We did not try to improve GDG on the last window in this work, but it is certainly worth investigating for future work.

Another possible improvement to GDG is automorphism ensemble decoding. In Appendix B, when decoding the 216 weight-two syndrome codewords, whose minimum weight solution consists of 14 VNs, GDG only succeeds in two of them. In data qubit decoding, the Tanner graph is highly symmetric and a lot of VNs have the same history, a better heuristic for guided decimation should be developed to break ties.

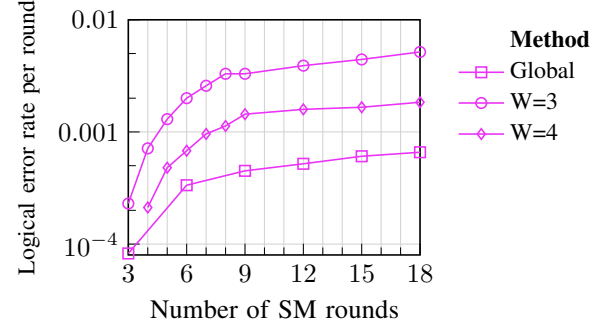


Fig. 7. Logical error rate per round when changing the number of syndrome measurement rounds for the $[[288, 12, 18]]$ code at physical error rate 0.005. BP+OSD-CS10 is used for global decoding. For the (3,1) or (4,1) sliding-window decoding, GDG is used on all but the last window and BP+OSD-CS10 on the last window.