Restart Belief: A General Quantum LDPC Decoder

Lorenzo Valentini, *Member, IEEE*, Diego Forlivesi, *Graduate Student Member, IEEE*, Andrea Talarico, *Graduate Student Member, IEEE*, and Marco Chiani, *Fellow, IEEE*

***Abstract*— Hardware-friendly quantum low-density parity-check (QLDPC) decoders are commonly built upon belief propagation (BP) processing. Yet, quantum degeneracy often prevents BP from achieving reliable convergence. To overcome this fundamental limitation, we propose the restart belief (RB) decoder, an iterative BP-based algorithm inspired by branch-and-bound optimization principles. We show that, for the considered QLDPC codes, the RB decoder achieves fast convergence and significantly improved decoding accuracy compared to existing decoders in the literature, while guaranteeing the error correction capability.**

***Index Terms*— Quantum computing, quantum error correcting codes, decoder, QLDPC codes, bivariate bicycle codes.**

I. Introduction

PRESERVING quantum information through quantum error correction (QEC) is essential for scalable quantum computing and reliable quantum memories [1]. Because quantum states are extremely fragile, QEC encodes logical qubits redundantly into many physical qubits and continuously detects and corrects errors without collapsing the state. In a typical QEC cycle, the system periodically measures stabilizer generators, identifies potential errors through syndrome extraction, and applies corrective operations to restore the quantum information [2]. Therefore, achieving practical fault-tolerant operation requires decoders that are not only accurate, but also extremely fast and hardware-friendly, capable of running efficiently on specialized platforms such as field-programmable gate arrays (FPGAs) or application-specific integrated circuits [3].

Among various QEC schemes, quantum low-density parity-check (QLDPC) codes are particularly promising due to their low-weight stabilizer generators, which reduce circuit complexity and error propagation while maintaining good distance properties. In particular, the Calderbank, Shor, and Steane (CSS) version of these codes is advantageous, as the separation between $\boldsymbol{Z}$-type and $\boldsymbol{X}$-type generators simplifies both the circuit design and the decoding process, while enabling many transversal encoded operations that are highly desirable for fault-tolerant quantum computation [4], [5]. These features make them attractive for large-scale fault-tolerant architectures where both qubit efficiency and reliable error suppression are critical. In this context, surface codes [6], structured instances of QLDPC codes, have received significant attention, with several specialized decoders such as minimum weight perfect matching (MWPM), union-find (UF), and bubble clustering (BC) [7], [8], [9], [10]. However, these decoders are highly specialized and tailored to the specific structure of surface codes. For this reason, researchers have actively explored belief propagation (BP)-based approaches for decoding general QLDPC codes.

In this direction, [11] proposed enhancing the BP algorithm by incorporating a post-processing stage based on ordered statistics decoding (OSD), resulting in the combined decoding strategy known as BP+OSD. However, this approach has certain drawbacks: the OSD step requires matrix inversion, which can be computationally expensive, and it does not always preserve the code distance. Subsequently, [12] introduced the decoder belief propagation based on guided decimation (BPGD), in which multiple instances of BP are executed iteratively until convergence is achieved. In each iteration where the decoder becomes trapped, the most reliable log-likelihood ratio (LLR) is fixed to guide the decoding process toward convergence. This algorithm is computationally more efficient, however, it provides a limited performance advantage compared to conventional BP. In [13] an enhanced version of the BPGD decoder with an integrated guessing algorithm is introduced. Recently, [3] introduced a novel decoding scheme, termed relay belief propagation (RelayBP), which aims to simultaneously achieve high decoding performance and low latency. The core concept involves running the BP algorithm and, in cases where convergence is not reached, restarting the process while updating the input LLRs using a damped version of the output LLRs from the previous iteration. This decoder has been shown to obtain good performance and decoding complexity for certain QLDPC codes such as the bivariate bicycle and the rotated surface ones.

In this letter, we continue the search for a general QLDPC decoder that balances performance and complexity. Our proposed decoder, hereafter named restart belief (RB), executes a first BP and its output LLRs are sorted. The minimum LLR is used to guide the decoder by applying an error to the corresponding qubit and updating the syndrome and the qubit LLR before the next BP. If the decoder still fails to converge, this process is repeated. Once convergence is achieved and a valid solution is obtained, the process is restarted using the second smallest LLR from the root BP, then the third, and so forth until a maximum is reached. In this way, the minimum LLRs of the root generate each a solution exploring a branch of the decision tree. To enhance the speed of the decoder we also include code distance-specific and defect-specific early termination strategies. In addition, it is worth noting that all the

Received 14 January 2026; accepted 14 February 2026. Date of publication 19 February 2026; date of current version 3 March 2026. This work is supported in part by PNRR MUR project PE0000023-NQSTI (Italy) financed by the European Union – Next Generation EU. The associate editor coordinating the review of this letter and approving it for publication was X. He. *(Corresponding author: Lorenzo Valentini.)*

The authors are with the Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi" and CNIT/WiLab, University of Bologna, 40136 Bologna, Italy (e-mail: lorenzo.valentini13@unibo.it; diego.forlivesi2@unibo.it; andrea.talarico3@unibo.it; marco.chiani@unibo.it).

Digital Object Identifier 10.1109/LCOMM.2026.3666352

branches BP, generating a solution, can be efficiently executed in parallel. To benchmark and validate our RB decoder we selected various QLDPC codes and compare the results with all the above mentioned QLDPC decoders based on BP.

II. Efficient Decoders for QLDPC Codes

The Pauli operators are denoted by $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$. A quantum error correcting code encoding k logical qubits $|\varphi\rangle$ into n physical qubits $|\psi\rangle$ with minimum distance d is represented as $[[n,k,d]]$, and can correct any error affecting up to $t = \lfloor (d-1)/2 \rfloor$ qubits. In the stabilizer formalism, the code is defined by $n-k$ independent, commuting generators $\boldsymbol{G}_i \in \mathcal{G}_n$, where $\mathcal{G}_n$ denotes the n-qubit Pauli group [14]. The subgroup generated by all $\boldsymbol{G}_i$ forms the stabilizer $\mathcal{S}$, and the code space $\mathcal{C}$ consists of all states $|\psi\rangle$ satisfying $\boldsymbol{S}|\psi\rangle = |\psi\rangle$ for every $\boldsymbol{S} \in \mathcal{S}$. Stabilizer generators correspond to measurements that leave the encoded state unchanged and are implemented using ancillary qubits. When an error $\boldsymbol{E} \in \mathcal{G}_n$ acts on a codeword, producing $\boldsymbol{E}|\psi\rangle$, the resulting binary syndrome vector $\boldsymbol{s}$ encodes commutation information: $s_i = 0$ if $\boldsymbol{G}_i$ commutes with $\boldsymbol{E}$ and $s_i = 1$ otherwise.

The code structure can be described by a bipartite graph with variable nodes, representing data-qubit errors, and check nodes, representing generator constraints. Decoding aims to find the most likely error configuration consistent with the observed syndrome. We consider CSS codes [4], [5], for which $\boldsymbol{X}$ and $\boldsymbol{Z}$ errors can be decoded independently. For clarity, we focus on $\boldsymbol{Z}$ error correction and consider only $\boldsymbol{X}$ stabilizer generators as check nodes. The corresponding parity-check matrix $\boldsymbol{H}_{\mathrm{x}}$ is formed by these generators, with one row per generator, and the decoder syndrome is denoted by $\boldsymbol{s}$. Decoding $\boldsymbol{X}$ errors proceeds analogously.

A. Belief Propagation

The BP algorithm iteratively exchanges probability messages between check and variable nodes to infer the locations of the most likely errors which are consistent with the measured syndrome. We denote as $m_{u_i \to v_j}$ the message from check node u_i to variable node v_j, and as $m_{v_j \to u_i}$ the message from a variable node v_j to a check node u_i. Also, we denote by $\mathcal{N}(u_i)$ the set of all neighboring variable nodes connected to the check node u_i. Analogously, $\mathcal{N}(v_j)$ is the set of check nodes neighboring the variable node v_j.

Given a check node i and a variable node j, the check-to-variable messages according to the scaled min-sum approach are computed as [15]

$$m_{u_i \to v_j} = (-1)^{s_i} \alpha \left[\prod_{v' \in \mathcal{N}(u_i) \setminus v_j} \operatorname{sign}\left(m_{v' \to u_i}\right) \right] \min_{v' \in \mathcal{N}(u_i) \setminus v_j} \left| m_{v' \to u_i} \right| \tag{1}$$

where α is a scaling factor, and s_i represents the i-th syndrome bit. The variable-to-check messages are computed as [15]

$$m_{v_j \to u_i} = p_j^{\text{in}} + \sum_{u' \in \mathcal{N}(v_j) \setminus u_i} m_{u' \to v_j} \tag{2}$$

where p_j^{in} is the input LLR computed with the channel prior information as $p_j^{\text{in}} = \log((1-p)/p)$, with p the physical error rate of the channel.

Next, the output LLR for each variable node is [15]

$$p_j^{\text{out}} = p_j^{\text{in}} + \sum_{u' \in \mathcal{N}(v_j)} m_{u' \to v_j}. \tag{3}$$

Finally, a hard decision is applied to determine a likely error as $\hat{e}_j = \frac{1}{2} - \frac{1}{2}\operatorname{sign}(p_j^{\text{out}})$, which is verified to ensure that the proposed error configuration is consistent with the observed syndrome. In the following we denote as T the number of BP iterations, i.e., the number of times (1) and (2) are iterated.

B. Ordered Statistic

The BP algorithm may fail to converge for quantum codes due to degeneracy, when multiple error configurations are equally likely and probability mass is split among them [16]. To address this, OSD is applied as a post-processing step, refining the BP estimate to produce a syndrome-compliant error configuration and ensuring reliable convergence [11]. This algorithm leverages the BP output LLR to construct an invertible, full-rank matrix $\boldsymbol{H}_s$ of size $r \times r$, which is then used to recover an estimated error vector $\hat{\mathbf{e}}_s \in \mathbb{F}_2^r$ from the syndrome $\mathbf{s}$ via $\hat{\mathbf{e}}_s = \boldsymbol{H}_s^{-1}\mathbf{s}$. The remaining qubit errors, associated to the vector $\hat{\mathbf{e}}_t \in \mathbb{F}_2^{n-r}$, are set to $\mathbf{0}$. The matrix $\boldsymbol{H}_s$ is formed by reordering the columns of the parity-check matrix $\boldsymbol{H}_{\mathrm{x}}$ according to decreasing qubit error probability. Columns are added sequentially, retaining only those that increase the rank; discarded columns are collected in $\boldsymbol{H}_t$. This process continues until $\boldsymbol{H}_s$ reaches full rank, yielding a square, invertible matrix, while $\boldsymbol{H}_t$ has dimensions $(n-r) \times r$.

Higher-order versions (OSD-1, OSD-2) refine the estimation by including qubits that were previously excluded from the matrix construction. This is done by setting $\hat{\mathbf{e}}_t$ to be different from $\mathbf{0}$ and computing the total error as [17]

$$\hat{\mathbf{e}}_{[s,t]} = \left(\hat{\mathbf{e}}_s + \boldsymbol{H}_s^{-1}\boldsymbol{H}_t\hat{\mathbf{e}}_t,\ \hat{\mathbf{e}}_t\right). \tag{4}$$

For OSD-1, (4) is evaluated for all weight-one configurations of $\hat{\mathbf{e}}_t$. For OSD-2, (4) is evaluated for all weight-two configurations restricted to the first λ bits of $\hat{\mathbf{e}}_t$, where λ is a user-defined parameter. Among all possible solutions, the one with the lowest Hamming weight is selected.

C. Belief Propagation Based on Guided Decimation

As a variant of the BP algorithm, BPGD was proposed in [12]. This decoder executes successive BP instances, where the subsequent ones adopt as input LLRs the output LLRs of the previous ones, except for the j-th LLR corresponding to the largest absolute value which is set accordingly to

$$p_j^{\text{in}} = \begin{cases} \infty, & \text{if } p_j^{\text{out}} > 0, \\ -\infty, & \text{if } p_j^{\text{out}} \leq 0. \end{cases} \tag{5}$$

Each BP instance run T BP iterations. The algorithm terminates if a valid solution is found or all qubits are decimated.

Algorithm 1 Restart Belief

input : t, ξ, $\boldsymbol{s}$, $\boldsymbol{H}_{\mathrm{x}}$;
output: estimated error;

$\boldsymbol{p}^{\mathrm{in}} \leftarrow \log\left((1-p)/p\right)$; $w_{\min} \leftarrow \infty$; $\hat{\boldsymbol{s}} \leftarrow \boldsymbol{s}$; $\boldsymbol{e}_{\min} \leftarrow \boldsymbol{0}$;
$[\hat{\boldsymbol{e}}, \boldsymbol{p}^{\mathrm{out}}] \leftarrow \mathtt{beliefProp}(\boldsymbol{p}^{\mathrm{in}}, T_{\mathrm{root}})$;
if $\boldsymbol{H}_{\mathrm{x}}\hat{\boldsymbol{e}} = \boldsymbol{s}$ **then**
  if $\mathtt{w}(\hat{\boldsymbol{e}}) \leq t$ or $\mathtt{w}(\boldsymbol{s})/\xi > t$ **then return** $\hat{\boldsymbol{e}}$;
  $\boldsymbol{e}_{\min} \leftarrow \hat{\boldsymbol{e}}$; $w_{\min} \leftarrow \mathtt{w}(\hat{\boldsymbol{e}})$;
$\mathcal{I} \leftarrow \mathtt{sort}(\boldsymbol{p}^{\mathrm{out}})$;
forall $i \in 1, \ldots, \eta$ **do**
  $\boldsymbol{p}^{\mathrm{in}} \leftarrow \log((1-p)/p)$; $\bar{\boldsymbol{e}} \leftarrow 0$;
  $\bar{\boldsymbol{e}}(\mathcal{I}[i]) \leftarrow 1$;
  forall $j \in 1, \ldots, t-1$ **do**
    $\hat{\boldsymbol{s}} \leftarrow \hat{\boldsymbol{s}} \oplus (\boldsymbol{H}_{\mathrm{x}}\bar{\boldsymbol{e}})$;
    $\boldsymbol{p}^{\mathrm{in}} \leftarrow \mathtt{fixLLR}(\boldsymbol{p}^{\mathrm{in}}, \bar{\boldsymbol{e}})$;
    $[\hat{\boldsymbol{e}}, \boldsymbol{p}^{\mathrm{out}}] \leftarrow \mathtt{beliefProp}(\boldsymbol{p}^{\mathrm{in}}, T_{\mathrm{branch}})$;
    if $\boldsymbol{H}_{\mathrm{x}}\hat{\boldsymbol{e}} = \hat{\boldsymbol{s}}$ **then break**;
    $\bar{\boldsymbol{e}}[\mathtt{argmin}(\boldsymbol{p}^{\mathrm{out}})] \leftarrow 1$;
  $\hat{\boldsymbol{e}} \leftarrow \hat{\boldsymbol{e}} \oplus \bar{\boldsymbol{e}}$;
  if $\boldsymbol{H}_{\mathrm{x}}\hat{\boldsymbol{e}} = \boldsymbol{s}$ **then**
    if $\mathtt{w}(\hat{\boldsymbol{e}}) \leq t$ or $\mathtt{w}(\boldsymbol{s})/\xi > t$ **then return** $\hat{\boldsymbol{e}}$;
    if $\mathtt{w}(\hat{\boldsymbol{e}}) < w_{\min}$ **then**
      $\boldsymbol{e}_{\min} \leftarrow \hat{\boldsymbol{e}}$; $w_{\min} \leftarrow \mathtt{w}(\hat{\boldsymbol{e}})$;
return $\boldsymbol{e}_{\min}$;

D. Relay Belief Propagation

The RelayBP algorithm has been recently proposed, incorporating memory across successive BP instances [3]. If the first BP instance fails to converge, the subsequent instance utilizes the output LLR probabilities from the previous run when computing the input channel probabilities as

$$p_{\ell}^{\mathrm{in}} = (1-\gamma)\, p_{0}^{\mathrm{in}} + \gamma\, p_{(\ell-1)}^{\mathrm{out}} \tag{6}$$

where ℓ denote the iteration step and γ is drawn from a uniform distribution over the interval $\left[\gamma_c - \frac{\gamma_w}{2}, \gamma_c + \frac{\gamma_w}{2}\right]$, with γ_c and γ_w as input parameters. These parameters can be adjusted according to the specific quantum code in order to optimize the decoding performance. In general, the procedure is repeated until S candidate solutions are obtained or until the maximum number of BP instances (or leg length) is reached. Among the obtained candidates, the solution with the minimal weight is then selected for error correction. As suggested in [3], the first leg employs T_0 iterations, while all the subsequent legs run $T_r \leq T_0$ iterations.

III. Restart Belief

In this section, we describe our proposal for a general CSS QLDPC decoder. In the following, we detail the decoding procedure for $\boldsymbol{Z}$-type Pauli errors. The same strategy applies to $\boldsymbol{X}$-type errors, with $\boldsymbol{Z}$ and $\boldsymbol{X}$ interchanged. For clarity, the proposed method is summarized in Algorithm 1, while Fig. 1 provides a schematic illustration to help guide the reader through the explanation.

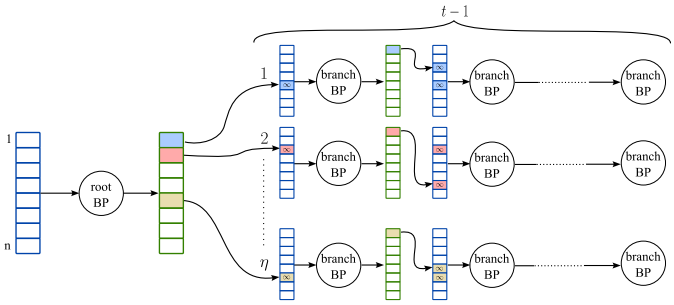

Fig. 1. Schematic representation of a restart belief decoding instance. To highlight the restarting approach: green vectors denote sorted LLRs, while blue vectors refer to the initial channel LLRs.

A. Decoder Description

As discussed in Section II, the degeneracy of quantum codes often prevents the convergence of the BP decoder. However, in most cases, we observe that the data qubits associated with the lowest LLRs correspond to the erroneous qubits. For this reason, we start with an initial BP instance, denoted in the following as the root BP, executing T_{root} BP iterations. In Algorithm 1, we employ the function $\mathtt{beliefProp}(\boldsymbol{p}^{\mathrm{in}}, T)$, implementing a BP instance with input LLRs $\boldsymbol{p}^{\mathrm{in}}$, T BP iterations, and returning the output LLRs $\boldsymbol{p}^{\mathrm{out}}$ along with the estimated error vector $\hat{\boldsymbol{e}}$. Note that, if the BP decoder does not converge, the estimated error vector is set to all zeros. At this point, we check for early termination. If $\hat{\boldsymbol{e}}$ is consistent with the original syndrome, we found a valid solution. However, this could not be the minimum weight (MW) solution. For this reason, we add a distance-specific and a defect-specific early termination rule. With the distance-specific rule we check if $\mathtt{w}(\hat{\boldsymbol{e}}) \leq t$ using the weight function $\mathtt{w}$. If this is the case we are sure to have found the MW solution. Conversely, the defect-specific condition triggers early termination if decoder detects that the error weight exceeds t. Specifically, we define as ξ the maximum column weight. In other words, ξ represents the maximum number of $\boldsymbol{X}$ parity checks that can anticommute per $\boldsymbol{Z}$ error. Hence, if $\mathtt{w}(\boldsymbol{s})/\xi > t$, we can conclude with certainty that the original error has weight greater than t. In this case, even if the weight of the obtained solution exceeds t, we accept it as the final solution.

In case $\hat{\boldsymbol{e}}$ is not consistent with the original syndrome or none of the termination criteria are satisfied, the algorithm proceeds by sorting the output LLRs in ascending order. The set $\mathcal{I}$ contains the indices of the reordered LLRs using the function $\mathtt{sort}$. Then, we start $\eta \leq n$ parallelizable searches. At the start of the i-th search (or branch) we apply a $\boldsymbol{Z}$ error on the qubit with the i-th lowest output LLR. The temporary error guess of the i-th search is stored in $\bar{\boldsymbol{e}}$. We accordingly update the syndrome as $\hat{\boldsymbol{s}} = \hat{\boldsymbol{s}} \oplus (\boldsymbol{H}_{\mathrm{x}}\bar{\boldsymbol{e}})$, where $\hat{\boldsymbol{s}}$ is a copy of the original syndrome $\boldsymbol{s}$. In addition, using the function $\mathtt{fixLLR}$, we set the input LLRs to ∞ in the location specified by $\bar{\boldsymbol{e}}$. Consequently, another BP instance is executed with T_{branch} BP iterations. If this instance of BP does not converge, a $\boldsymbol{Z}$ error is imposed on the qubit with the lowest output LLR. The syndrome and the input LLRs are then updated accordingly, as described above, and a new instance of BP is executed. This procedure is repeated iteratively until either the BP converges

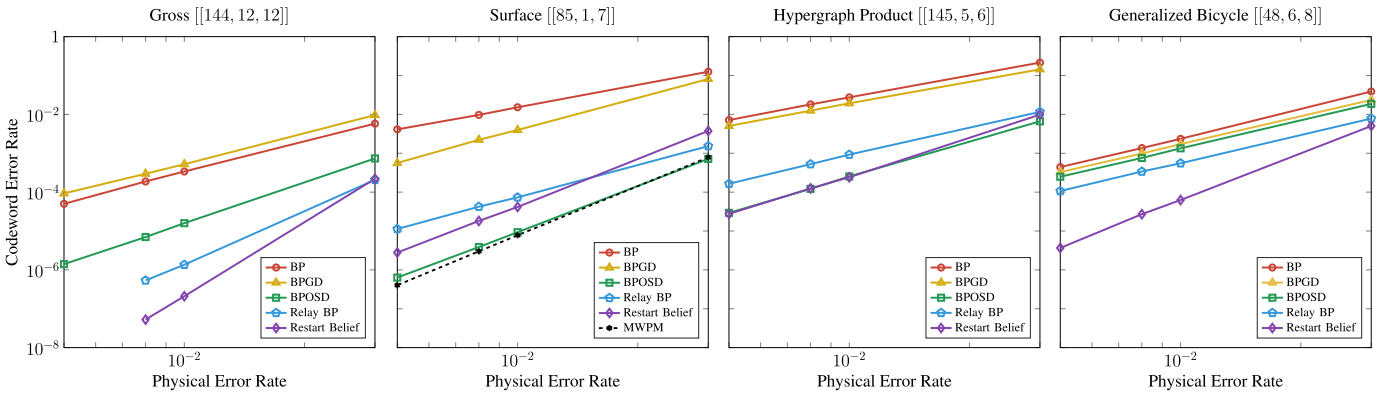

Fig. 2. Codeword error rate vs. physical error rate varying codes and decoders. QLDPC code classes from left to right: bivariate bicycle, topological, hypergraph product, and generalized bicycle.

or $t = \lfloor (d-1)/2 \rfloor \boldsymbol{Z}$ errors have been inserted into $\bar{\boldsymbol{e}}$. Next, we check if the final estimated error $\hat{\boldsymbol{e}} = \hat{\boldsymbol{e}} \oplus \bar{\boldsymbol{e}}$ is consistent with the original syndrome. In case $\hat{\boldsymbol{e}}$ is consistent, we also check the same conditions used for the root solution, storing the current solution if it has the lowest weight so far and none of the termination criteria are met. In case $\hat{\boldsymbol{e}}$ is not consistent or none of the termination criteria are satisfied, we proceed with the exploration of the next branch, until the maximum number of searches η have been explored. Finally, if none of the early termination conditions are ever met, the decoder outputs the stored solution.

Note that, the parameter η can be viewed as a trade-off lever between performance and latency. In the following, we fine-tune this parameter to ensure that our decoder can systematically correct all error patterns of weight up to t. This systematic nature in our decoding strategy, which provides performance guarantees, is one of its key strengths compared to more stochastic approaches, such as RelayBP.

B. Complexity Analysis

The RB decoder performs a single root BP instance and at most η branch searches, where each branch consists of at most $t-1$ BP instances. The root instance runs T_{root} iterations, while each branch instance runs T_{branch} iterations. Considering a single branch, we have a computational cost of $\mathcal{O}((t-1)T_{\text{branch}}n) \approx \mathcal{O}(tT_{\text{branch}}n)$, due to the fact each BP iteration has a complexity of $\mathcal{O}(n)$. Hence, considering also the root BP instance, the overall worst-case complexity of the RB decoder is $\mathcal{O}((\eta t T_{\text{branch}} + T_{\text{root}})n)$. This scaling is similar to that of conventional BP-based iterative decoders, while remaining considerably lower than that of OSD-0, which exhibits a cubic complexity of $\mathcal{O}(n^3)$ [11]. It is worth noting that the RB decoder naturally supports hardware parallelization, since the η outer branching rounds can be evaluated independently and mapped to parallel processing units, reducing the effective worst-case complexity to $\mathcal{O}((tT_{\text{branch}} + T_{\text{root}})n)$.

IV. Numerical Results

In this section, we compare the decoders discussed in Section II with the proposed decoder described in Section III through Monte Carlo simulations. The comparison is carried out considering four different QLDPC codes to show also the decoder flexibility. The set of codes used in this comparison comprises a bivariate bicycle code, a topological code, an hypergraph product code, and a generalized bicycle code, thereby enabling the exploration of a broad spectrum of QLDPC structures. In particular, as topological code we adopt the $[[85, 1, 7]]$ surface code [6]. As hypergraph product code we adopt the $[[145, 5, 6]]$ code constructed from an augmented classical repetition code [17]. As a bivariate bicycle code we adopt the $[[144, 12, 12]]$ gross code [18], [19]. As a generalized bicycle code we adopt the $[[48, 6, 8]]$ code from [20]. For each physical error rate, the simulations are executed until 100 decoding failures are observed, ensuring statistically meaningful estimates of the codeword error rate. The source code implementation is available in our repository QBelief [21].

For the Relay decoder, we use a leg length of 301, $T_0 = 80$, $T_r = 60$, $S = 5$, and $\alpha = 1$, as provided in [3]. Additionally, we tested the best parameter values reported in [3] for the rotated surface and gross codes, as well as an additional value obtained by averaging the two. It turns out that, for all tested codes, the pair $[-0.24, 0.66]$ provides the best performance among our three possibilities. For all the remaining decoders, we use $\alpha = 1 - 2^{-N_{\text{iter}}}$, where N_{iter} is the current iteration [15], [17]. For the BP+OSD, we employ an OSD-2 with $\lambda = 10$ as suggested in [17]. For the BP+OSD, the BPGD and the baseline BP, we set $T = 50$. Also, for the BPGD decoder we set $p_j^{\text{in}} = 25$ during decimation, as suggested in [12]. For the RB decoder we set $T_{\text{root}} = 50$ and $T_{\text{branch}} = 10$. The parameter η has been fine-tuned by exhaustive search optimization for each of the tested codes to guarantee the error correction capability t. Note that, for different choices of α, T_{root}, and T_{branch}, the optimum values of η may differ. In particular, η values are listed in Table I for each code.

In Fig. 2, we report the codeword error rate of several code-decoder pairs over a depolarizing channel versus the depolarizing channel physical error rate. From the figure, it can be observed that the RB decoder outperforms the decoders based on BP, BPGD, and RelayBP. Regarding the $[[85, 1, 7]]$ surface code, our proposal exhibits a performance gap compared to the BP+OSD. As shown in [22], this constant gap is due to the fact that the RB decoder correct less error pattern of weight $t + 1$. It is worth noting that the slope of the two curve is the same of the MWPM decoder, meaning

TABLE I

AVERAGE BP ITERATIONS

Code	Decoder	Configuration	$n_e = 1$	$n_e = 2$	$n_e = 3$	$n_e = 4$	$n_e = 5$	$n_e = 6$	$n_e = 7$	$n_e = 8$	$n_e = 9$	$n_e = 10$	$n_e = 11$	$n_e = 12$
$[[48, 6, 8]]$	**Restart Belief**	$\eta = 48$	1.000	5.085	35.44	812.3	792.9	859.1	799.7	842.9				
	Relay BP		91.66	191.8	478.3	3361	13831	14458	14633	14848				
$[[144, 12, 12]]$	**Restart Belief**	$\eta = 35$	1.000	1.102	1.577	2.545	3.249	370.7	275.3	64.57	66.22	39.22	77.50	138.6
	Relay BP		56.28	70.49	91.09	111.4	133.5	178.3	242.1	291.1	372.9	531.9	972.6	2065
$[[85, 1, 7]]$	**Restart Belief**	$\eta = 8$	2.000	4.628	11.03	41.36	50.65	72.99	91.75	109.2				
	Relay BP		31.42	67.52	145.3	222.1	341.1	445.1	585.0	738.1				
$[[145, 5, 6]]$	**Restart Belief**	$\eta = 6$	1.891	3.484	24.31	15.52	22.84	33.33						
	Relay BP		45.47	69.78	108.8	158.8	203.4	243.1						

that both decoders protect the codeword within the code distance. Regarding the gross code, we highlight that the RB decoder outperforms RelayBP on the gross code, while the BP+OSD is not able to preserve the code distance. Regarding the $[[145, 5, 6]]$ hypergraph product code, the BP+OSD, the RelayBP, and the RB perform practically the same, with only a slight gap for the RelayBP. Finally, regarding the $[[48, 6, 8]]$ generalized bicycle, we observe a significant performance advantage when adopting the RB decoder.

Table I reports the average number of BP iterations used by the two decoders that offer the best compromise between hardware complexity and performance: the RB and the RelayBP. These numbers include all BP iterations required for the execution of the algorithm, both the root and branches for the RB and both the first BP and the legs of the RelayBP. The table is generated by varying n_e, the number of introduced $\boldsymbol{Z}$ errors, to illustrate how computational cost depends on the error weight handled by the decoder. The η parameters listed in Table I have been selected as the minimum value that guarantees the error correction capability. To this end, we perform exhaustive-search simulations, starting from $\eta = 1$ and incrementally increasing its value until all such errors of weight less than t are correctly decoded. On the other hand, the non-deterministic nature of the RelayBP leads to variability in the decoding outcomes, causing certain error patterns within the code distance to be occasionally misdecoded. For all codes except to the gross code, the RB decoder consistently proves to be faster than the RelayBP decoder. In the case of the gross code, the RB decoder requires a higher number of BP iterations for errors of weight $n_e = t + 1$ and $n_e = t + 2$. This is because the defect-specific criterion cannot uniquely identify such error patterns.

V. CONCLUSION

In this work, we introduce a novel decoder for QLDPC codes. Across all tested quantum codes, our results show that the proposed RB decoder achieves the lowest logical error rate while maintaining a computational complexity comparable to other BP-based decoders reported in the literature. This work focuses on the evaluation under the depolarizing channel to benchmark the intrinsic decoding performance. The system-level analysis accounting circuit-level noise is left for future work. As additional features, we highlight that the RB decoder is parallelizable, easy-to-tune, and deterministic. For the code under consideration, the simulation analyses, complemented with exhaustive searches, show that our decoder guarantees the error correction capability.

REFERENCES

[1] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A, Gen. Phys.*, vol. 52, no. 4, pp. R2493–R2496, Oct. 1995.

[2] D. Gottesman, "Class of quantum error-correcting codes saturating the quantum Hamming bound," *Phys. Rev. A, Gen. Phys.*, vol. 54, no. 3, pp. 1862–1868, Sep. 1996.

[3] T. Müller et al., "Improved belief propagation is sufficient for real-time decoding of quantum memory," 2025, *arXiv:2506.01779*.

[4] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Phys. Rev. A, Gen. Phys.*, vol. 54, no. 2, pp. 1098–1105, Aug. 1996.

[5] S. Andrew, "Multiple-particle interference and quantum error correction," *Proc. Roy. Soc. London.*, vol. 452, no. 1954, pp. 2551–2577, 1996.

[6] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," 1998, *arXiv: quant-ph/9811052*.

[7] O. Higgott and C. Gidney, "Sparse blossom: Correcting a million errors per core second with minimum-weight matching," *Quantum*, vol. 9, p. 1600, Jan. 2025.

[8] N. Delfosse and N. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, vol. 5, p. 595, Dec. 2021.

[9] D. Forlivesi, L. Valentini, and M. Chiani, "Spanning tree matching decoder for quantum surface codes," *IEEE Commun. Lett.*, vol. 28, no. 7, pp. 1509–1513, Jul. 2024.

[10] D. Forlivesi, L. Valentini, and M. Chiani, "Bubble clustering decoder for quantum topological codes," *IEEE Trans. Commun.*, vol. 73, no. 10, pp. 8470–8483, Oct. 2025.

[11] P. Panteleev and G. Kalachev, "Degenerate quantum LDPC codes with good finite length performance," *Quantum*, vol. 5, p. 585, Nov. 2021.

[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," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2024, pp. 2478–2483.

[13] A. Gong, S. Cammerer, and J. M. Renes, "Toward low-latency iterative decoding of QLDPC codes under circuit-level noise," 2024, *arXiv:2403.18901*.

[14] D. Gottesman, "An introduction to quantum error correction and fault-tolerant quantum computation," in *Proc. Symp. Appl. Math.*, vol. 68, 2010, pp. 13–58.

[15] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[16] D. Poulin and Y. Chung, "On the iterative decoding of sparse quantum codes," *Quantum Inf. Comput.*, vol. 8, no. 10, pp. 986–1000, Nov. 2008.

[17] 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, no. 4, Dec. 2020, Art. no. 043423.

[18] 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," *Nature*, vol. 627, no. 8005, pp. 778–782, Mar. 2024.

[19] T. J. Yoder et al., "Tour de gross: A modular quantum computer based on bivariate bicycle codes," 2025, *arXiv:2506.03094*.

[20] A. A. Kovalev and L. P. Pryadko, "Quantum Kronecker sum-product low-density parity-check codes with finite rate," *Phys. Rev. A, Gen. Phys.*, vol. 88, no. 1, Jul. 2013, Art. no. 012311.

[21] (2025). *Qbelief*. [Online]. Available: https://github.com/DiegoForlivesi/QBelief

[22] D. Forlivesi, L. Valentini, and M. Chiani, "Performance analysis of quantum error-correcting codes via Williams identities," *Quantum*, vol. 9, p. 1950, Dec. 2025.

Open Access funding provided by 'Alma Mater Studiorum - Università di Bologna' within the CRUI CARE Agreement