LIFTDIFF: GUIDED DISCRETE DIFFUSION FOR QUANTUM CODE CONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

The realization of fault-tolerant quantum computation relies on Quantum Error-Correcting Codes (QECC). Of particular interest are codes that combine a high encoding rate with sufficient distance. Existing algebraic constructions, such as lifted product (LP) codes, build a large quantum low-density parity-check (qLDPC) code from a classical base matrix. In this work we introduce LiftDiff, a generative approach to the discovery of QECC. A discrete denoising diffusion model is trained to generate base matrices over the group algebra $\mathbb{F}_2[\mathbb{Z}_\ell]$, which are then expanded into LP codes. The reverse diffusion process is guided by an energy function that evaluates quantum code properties, including girth, degree, sparsity and decoding performance, steering the sampler toward matrices that yield LP codes with favorable parameters. Our method discovers codes with parameters such as $[[169, 13, 11]]$, $[[143, 15, 9]]$, and $[[260, 24, 10]]$, including a $[[247, 19, 14]]$ code that achieves the highest kd^2/n among the baselines. Moreover, several discovered codes outperform the Gross code across a broad range of the tested physical error rates under both code-capacity and circuit-level noise.

1 INTRODUCTION

Quantum computing harnesses the laws of quantum mechanics to perform computation, giving rise to a fundamentally different computational paradigm from classical computing. This paradigm has the potential to provide substantial computational advantages for problems that are difficult to solve using classical methods (Shor, 1997; Grover, 1996; Gidney & Ekerå, 2021; Feynman et al., 2018). Quantum systems are, however, exceptionally noisy. The qubit, the elementary unit of quantum information, is coupled to its environment, and the resulting decoherence destroys the superpositions and phase relations on which the computation depends. Errors therefore accumulate before a computation of useful length can be completed. Quantum Error Correction (QEC) (Calderbank & Shor, 1996) is thus indispensable for the realization of a quantum computer. Shor's first QEC scheme (Shor, 1995) demonstrated that these obstacles can be overcome, and that quantum information can be protected despite the impossibility of copying it and the continuous nature of the errors that afflict it. Considerable progress has been made since through the stabilizer formalism (Gottesman, 1997). Within the stabilizer framework, Calderbank-Shor-Steane (CSS) codes (Steane, 1996; Calderbank & Shor, 1996) form an important class of QECC constructed from two parity check matrices satisfying an orthogonality condition. Their structure separates the correction of bit-flip and phase-flip errors. Among the QEC schemes proposed, the surface code (Kitaev, 1997; Bravyi & Kitaev, 1998; Kitaev, 2003; Dennis et al., 2002) has received the most attention. It tolerates a comparatively high physical error rate, and efficient decoders for it are available. Its drawback is the encoding rate. A surface code patch encodes a single logical qubit into a number of physical qubits that grows quadratically with the distance, so its encoding rate vanishes as the protection is increased. This has motivated the search for codes that keep the parity checks sparse while encoding many logical qubits in the same block. Classical LDPC codes combine sparse checks with efficient message-passing decoding and performance close to capacity, and recent years have seen sustained effort to create qLDPC codes (Breuckmann & Eberhardt, 2021), where the checks must in addition commute. The Hypergraph Product (HGP) (Tillich & Zemor, 2014) provides a way to construct a qLDPC code using the parity check matrix of two classical codes. The LP construction replaces the binary entries of the base matrix with elements of a group algebra (Panteleev & Kalachev, 2022), allowing the constructed codes to achieve high encoding rate and distance. Among the most promising qLDPC constructions are

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the Bivariate Bicycle (BB) (Bravyi et al., 2024) codes, whose CSS structure is defined through a pair of polynomials in the ring $\mathbb{F}_2[x, y]/(x^\ell - 1, y^m - 1)$. Machine learning has been applied to a range of problems in the domain of quantum computing, such as quantum error decoding (Wang & Tang, 2024; Bausch et al., 2024; Senior et al., 2026; Chamberland et al., 2026) and the design of QECC (Olle et al., 2024; Su et al., 2025; Liu & Marquardt, 2026; Yan et al., 2026b; Cruz-Benito et al., 2026). Several approaches have been used to search the space of QECC. However, the discrete and combinatorial nature of the design space, together with the costly evaluation of the resulting codes, makes efficiently navigating this space challenging, particularly for larger constructions. Generative models offer another approach to this problem. A generative model can learn the underlying structure of the design space and produce new candidate constructions.

In this work, we introduce LiftDiff, a generative approach to the discovery of QECC. Our method employs a discrete diffusion model trained on classical base matrices, which are subsequently used to construct quantum codes through LP. During inference, we guide the diffusion process using a set of energy functions designed to promote desirable code properties, steering the generation toward promising quantum code constructions. Our contributions are as follows:

- We introduce, to the best of our knowledge, the first discrete diffusion framework for the generative construction of novel QECC.
- We develop an energy-based guidance mechanism that operates throughout the reverse diffusion process, using differentiable quantum code properties to steer the generation toward base matrices with desirable properties after lifting.
- LiftDiff can generate QECC with diverse combinations of code parameters, including $[[143, 15, 9]]$, $[[260, 24, 10]]$, and $[[169, 13, 11]]$, as well as a $[[247, 19, 14]]$ code that achieves the highest kd^2/n among all baselines for a check weight ≤ 8.

2 Related Work

Two major topics of research in QEC (Roffe, 2019) are construction of quantum codes and the development of algorithms for decoding them, with recent work increasingly exploring machine learning as a tool for both tasks (Wang & Tang, 2024; Alexeev et al., 2025; Krenn et al., 2023; Yan et al., 2026a). The construction of quantum codes has traditionally relied on algebraic and combinatorial methods, leading to a variety of code families with different trade-offs between rate, distance, and structural properties (Bravyi et al., 2024; Tillich & Zemor, 2014; Panteleev & Kalachev, 2022; Breuckmann & Eberhardt, 2021; Kasai, 2026; Zhao et al., 2026; Breuckmann & Terhal, 2016). In terms of decoding, a wide range of decoders have been developed, such as Minimum Weight Perfect Matching, Union Find and Belief Propagation with Ordered Statistics Decoding (BP-OSD) (Edmonds, 1965; Fowler, 2014; Delfosse & Nickerson, 2021; Panteleev & Kalachev, 2021), as well as their variants (Wu et al., 2025; Peled et al., 2026; Kwak et al., 2026). More recently machine learning was also applied to develop neural decoders, spanning from Convolutional Neural Networks (CNN) (Gu et al., 2026), Graph Neural Networks (GNN) (Lange et al., 2025) to Transformers (Blue et al., 2026; Zenati & Nachmani, 2026; Choukroun & Wolf, 2023). Code Design was also explored using machine learning techniques, starting from reinforcement learning (Olle et al., 2024; Nautrup et al., 2019; Su et al., 2025; Sella et al., 2026), bayesian optimization (Chengyu et al., 2026) to the usage of Large Language Models (LLMs) (Cruz-Benito et al., 2026; Yan et al., 2026b).

3 Background

Quantum Foundations. A qubit is the basic unit of quantum information and can be described by a state: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $\alpha, \beta \in \mathbb{C}$ are probability amplitudes satisfying $|\alpha|^2 + |\beta|^2 = 1$. An n-qubit system is described by a state in a 2^n-dimensional Hilbert space. QECC exploit this structure to encode logical information into a larger set of physical qubits and protect it from noise.
Stabilizer Formalism. The stabilizer formalism (Gottesman, 1997) encodes quantum information by constraining an n-qubit system to the simultaneous $+1$ eigenspace of a commuting set of Pauli operators within the Hilbert space $\mathcal{H}_2^n$. These operators are drawn from the n-qubit Pauli group $\mathcal{P}_n$, which consists of tensor products of single-qubit Pauli matrices. Errors manifest as operators that violate these symmetry constraints and can be identified through projective measurements

that do not disturb the encoded logical information. Concretely, an $[[n, k, d]]$ quantum code encodes k logical qubits into n physical qubits with distance d, the stabilizer code is specified by choosing $m = n - k$ independent generators $\{S_i\}_{i=1}^m$ subject to the constraint that the group generated by these operators does not contain $-I$. The associated logical code space is given by $C_S = \{ |\psi\rangle \in \mathcal{H}_2^n \mid S_i|\psi\rangle = |\psi\rangle, \forall i \in \{1, \ldots, m\}\}$. Measuring the stabilizer generators produces a binary syndrome that reflects the commutation relations between an error operator and the stabilizers, thereby enabling error identification while preserving the logical state, and a set of commuting stabilizer generators can be represented by a binary check matrix.

CSS Codes. CSS codes are a subclass of stabilizer codes in which the stabilizer generators are decoupled into X-type and Z-type checks. They can therefore be described by two binary matrices H_X and H_Z, which must satisfy $H_X H_Z^T = 0$ to ensure that all stabilizer generators commute. In this case, the number of encoded logical qubits is: $k = n - \text{rank}(H_X) - \text{rank}(H_Z)$

Lifted Product Codes. The LP construction (Panteleev & Kalachev, 2022) provides a systematic way of constructing CSS quantum codes from matrices over a group algebra. Let

$$M_1 \in R^{m_1 \times n_1}, \qquad M_2 \in R^{m_2 \times n_2}, \tag{1}$$

where R is a group algebra over $\mathbb{F}_2$. The construction combines M_1 and M_2 and their involutions to form the CSS check matrices

$$H_X = [M_1 \otimes I_{n_2} \mid I_{m_1} \otimes M_2^*], \qquad H_Z = [I_{n_1} \otimes M_2 \mid M_1^* \otimes I_{m_2}], \tag{2}$$

where M^* denotes the transpose together with the involution on the group algebra. By construction, the resulting matrices satisfy $H_X H_Z^* = 0$ and therefore define a CSS code. For the cyclic group algebra $R = \mathbb{F}_2[x]/(x^\ell - 1)$, each entry of the base matrices is a polynomial of degree less than ℓ, and the lifted representation replaces each polynomial by its corresponding $\ell \times \ell$ circulant binary matrix. In this work, we consider the self-product case $LP(M, M)$.

4 METHOD

4.1 OVERVIEW

We propose a novel framework for constructing QECCs, specifically LP codes. LiftDiff does not operate directly on parity-check matrices, instead, it operates on a base matrix $M \in R^{m \times n}$ over the group algebra $R = \mathbb{F}_2[\mathbb{Z}_\ell]$. The matrix M then defines a quantum code via $LP(M, M)$. The QECC search space is enormous, making the search for codes with specific properties challenging. Discrete diffusion is particularly well suited to this task for two reasons: (i) codes are binary, making discrete diffusion over binary instances a natural choice. (ii) The iterative nature of the diffusion process, together with its differentiable formulation, allows us to specify desired code properties and guide the diffusion process toward high-quality codes. Our pipeline consists of three stages:

(i) Training. A discrete denoising diffusion model is trained to invert a bit-flip corruption process on base matrices drawn from a progressive-edge-growth (PEG) construction. The model learns what a sparse, high-girth base matrix looks like.

(ii) Guided inference. The trained model is sampled in reverse, and at every denoising step the prediction is displaced along the gradient of an energy that scores the quantum code the current base matrix implies. The energy is computed by pushing the model's continuous prediction through the LP. This stage produces batches of base matrices. This process is illustrated in Figure 1

(iii) Screening and certification. Every sample is expanded, verified to commute, and given its exact n and k, a relatively cheap distance bound (Pryadko et al., 2022) is used to give a first indication of each code. The distance of the best codes is finally proven using Mixed-Integer Linear Programming (MILP) (Cruz-Benito et al., 2026).

4.2 THE GENERATIVE MODEL

Corruption process. Let $M_0 \in \{0, 1\}^{m \times n \times \ell}$ be a clean base matrix, written as a tensor whose last axis holds the ℓ coefficients of each cell's polynomial. The forward process is the discrete diffusion of D3PM (Austin et al., 2021) with a symmetric binary channel: at step t every bit is

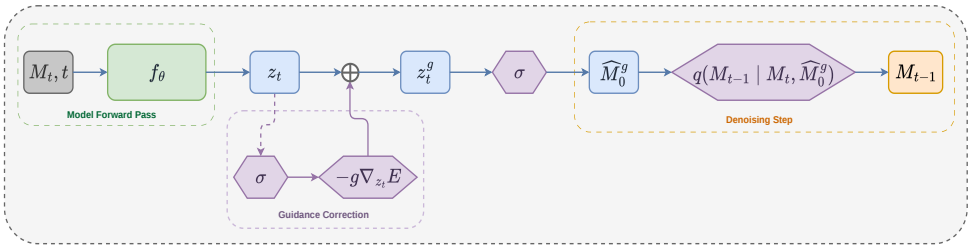


Figure 1: Overview of the inference process. The matrix M_t and timestep t are input to the model, producing logits z_t. The logits are passed through a sigmoid function and fed into the guidance mechanism, whose output adjusts the prediction to obtain z_t^g. The guided logits are passed through a second sigmoid function, and the resulting output is used to produce M_{t-1}.

flipped independently with probability β_t. Composing t steps gives a single Bernoulli channel,

$$M_t = M_0 \oplus \varepsilon_t, \qquad \varepsilon_t \sim \text{Bernoulli}(1-\bar{\alpha}_t)^{\otimes mn\ell}, \qquad \bar{\alpha}_t = \tfrac{1}{2}\Big(1 + \prod_{s \le t}(1-2\beta_s)\Big), \tag{3}$$

where $\bar{\alpha}_t$ is the probability that a given bit has survived unflipped. As $t \to T$, $\bar{\alpha}_t \to \frac{1}{2}$ and M_T is uniform noise. We use either a linear schedule on β_t or a cosine schedule on $\bar{\alpha}_t$ rescaled to $(\frac{1}{2}, 1]$, which is the reachable range for a binary symmetric channel.

4.3 Model Architecture

The backbone of our model is a message passing architecture utilizing cross-attention mechanisms over a bipartite graph (Vaswani et al., 2017). Let the base matrix be a tensor $M \in \mathbb{F}_2^{m \times n \times \ell}$, where each of the ℓ entries in a cell indicates whether the corresponding term appears in the polynomial. Given a noisy tensor M_t at diffusion timestep t, the model aims to predict the clean tensor M_0. The architecture initializes learnable node embeddings for the check and variable nodes, $Z_C^{(0)} \in \mathbb{R}^{m \times d}$ and $Z_V^{(0)} \in \mathbb{R}^{n \times d}$, where d is the hidden dimension. The scalar timestep t is mapped to a continuous time embedding $c \in \mathbb{R}^d$ via sinusoidal positional encodings (Vaswani et al., 2017) followed by a multi-layer perceptron (MLP). To incorporate the structural information of the noisy graph, M_t is collapsed along its ℓ-dimension to form a bipartite adjacency matrix $\tilde{M}_t \in \mathbb{R}^{m \times n}$. For each check node i and variable node j, the ℓ-dimensional vector $(M_t)_{i,j}$ is projected via a learned linear layer:

$$\tilde{M}_{t,i,j} = \mathbf{w}^T (M_t)_{i,j} + b, \tag{4}$$

where $\mathbf{w} \in \mathbb{R}^\ell$ and $b \in \mathbb{R}$. The core of the network consists of L bipartite attention blocks. In each block, node representations are first modulated by the time embedding c using Adaptive Layer Normalization (AdaLN). Specifically, c is projected via a weight matrix $W_1 \in \mathbb{R}^{d \times 4d}$ to generate scale and shift parameters $\gamma_C, \beta_C, \gamma_V, \beta_V \in \mathbb{R}^d$:

$$[\gamma_C, \beta_C, \gamma_V, \beta_V] = W_1(\text{SiLU}(c)), \tag{5}$$

which are used to normalize the check and variable node representations:

$$\hat{Z}_C = \text{LayerNorm}(Z_C) \odot (1+\gamma_C) + \beta_C, \quad \hat{Z}_V = \text{LayerNorm}(Z_V) \odot (1+\gamma_V) + \beta_V. \tag{6}$$

During the variable-to-check ($V \to C$) message passing phase, check nodes attend to variable nodes. To account for multi-head attention with H heads, let $d_h = d/H$ be the head dimension. For each head $h \in \{1, \dots, H\}$, the queries, keys, and values are explicitly computed from the normalized representations: $Q_C^{(h)} = \hat{Z}_C W_{Q,h}^C, K_V^{(h)} = \hat{Z}_V W_{K,h}^V, V_V^{(h)} = \hat{Z}_V W_{V,h}^V$ where $W_{Q,h}^C, W_{K,h}^V, W_{V,h}^V \in \mathbb{R}^{d \times d_h}$ are the weight matrices for head h, resulting in $Q_C^{(h)} \in \mathbb{R}^{m \times d_h}$ and $K_V^{(h)}, V_V^{(h)} \in \mathbb{R}^{n \times d_h}$. The structural bias is injected into the pre-softmax affinities of each head:

$$U_C^{(h)} = \text{Softmax}\left(\frac{Q_C^{(h)} (K_V^{(h)})^T}{\sqrt{d_h}} + \alpha^{(h)} \tilde{M}_t\right) V_V^{(h)}, \tag{7}$$

where $\alpha^{(h)} > 0$ is a time-dependent learned scalar specific to head h. The outputs $U_C^{(h)}$ from all H heads are concatenated to form the combined update $U_C \in \mathbb{R}^{m \times d}$. The node representations are then updated via an MLP and a residual connection:

$$Z_C^{\text{out}} = Z_C + \text{MLP}_C(U_C). \tag{8}$$

A symmetric operation updates the variable nodes ($C \to V$) using $\tilde{M}_t^T$ to yield Z_V^{out}. After L iterations and a final LayerNorm, the representations $Z_C^{(L)}$ and $Z_V^{(L)}$ are processed by an edge decoder. We employ an independent bilinear map for each polynomial coefficient index $k \in \{1, \dots, \ell\}$:

$$\text{Logits}_{i,j,k} = \frac{1}{\sqrt{d}} \left(Z_{C,i}^{(L)}\right)^T W^{(k)} Z_{V,j}^{(L)}, \quad W^{(k)} \in \mathbb{R}^{d \times d}. \tag{9}$$

Finally, a sigmoid activation $\sigma(\cdot)$ is applied to output the predicted tensor $\hat{M}_0 \in (0,1)^{m \times n \times \ell}$:

$$\widehat{M_0} = (\hat{M}_0)_{i,j,k} = \sigma(\text{Logits}_{i,j,k}) \tag{10}$$

Training objective. Training is the standard x_0-prediction objective, we draw M_0 from the data, draw t uniformly, corrupt, and ask the network to recover M_0. Since every entry is a bit, the reconstruction term is a binary cross-entropy over all $mn\ell$ coefficients. To this we add a degree term d_v to gain control over the final structure. The degree term is computed on the *support*, not on the coefficients, we collapse the polynomial axis by using a differentiable soft-OR, the probability that cell (i,j) is nonzero: $\widehat{S}_{ij} = 1 - \prod_k \big(1 - \widehat{M}_{0,ijk}\big)$. Thus, the final training objective is:

$$\mathcal{L} = \text{BCE}\big(\widehat{M_0},\, M_0\big) + \lambda_{\deg} \left(\big\|\text{col}(\widehat{S}) - d_v\big\|_{\text{mean}}^2 + \big\|\text{row}(\widehat{S}) - \tfrac{n d_v}{m}\big\|_{\text{mean}}^2 \right) \tag{11}$$

4.4 Guided inference

To generate new codes, and to optimize their properties, we guide the diffusion process along its whole trajectory. We define an energy E as a weighted sum of sub-energies, $E = \sum_i \lambda_i E_i$ where each E_i represents one property of the code that we wish to optimize and $\lambda_i \geq 0$ sets its strength. Unguided sampling draws the next state from the D3PM posterior using the network prediction. Writing $z_t = f_\theta(M_t, t)$ for the logits the network produces at step t, one reverse step is

$$z_t = f_\theta(M_t, t), \qquad \widehat{M_0} = \text{sigmoid}(z_t), \qquad M_{t-1} \sim q\big(M_{t-1} \mid M_t,\, \widehat{M_0}\big). \tag{12}$$

We modify equation 12 to

$$\begin{aligned} z_t &= f_\theta(M_t, t), & z_t^g &= z_t - g\, \nabla_{z_t} E\big(\text{sigmoid}(z_t)\big), \\ \widehat{M_0^g} &= \text{sigmoid}\big(z_t^g\big), & M_{t-1} &\sim q\big(M_{t-1} \mid M_t,\, \widehat{M_0^g}\big). \end{aligned} \tag{13}$$

where g is a tunable hyperparameter defined as the guidance scale. The guidance is performed in the logit space rather than probabilities. Logits are unconstrained unlike probabilities. Nothing about the corruption process is modified. The forward process, the schedule and the posterior q are exactly those of Section 4.2, the guidance enters only through $\widehat{M_0}$ which the posterior is conditioned on.

Energy Terms The energy scores the code implied by a continuous base matrix. Two objects are used, and each term below is defined on one of them explicitly. Pushing $\widehat{M_0} \in [0,1]^{m \times n \times \ell}$ through the LP with the mod-2 reduction dropped gives real-valued check matrices H_X, H_Z, applying the lift alone gives the *unfolded base matrix*

$$\mathcal{M} \;=\; unfold\big(\widehat{M_0}\big) \;\in\; \mathbb{R}^{\bar{m} \times \bar{n}}, \qquad \bar{m} = m\ell, \quad \bar{n} = n\ell, \tag{14}$$

the Tanner matrix of the classical base graph, whose rows are check nodes and whose columns are variable nodes. Parities of real numbers are taken with

$$\text{par}(s) \;=\; \tfrac{1}{2}\big(1 - \cos(\pi s)\big), \tag{15}$$

which returns $s \bmod 2$ whenever s is an integer and interpolates smoothly in between, so a syndrome can be formed from real-valued matrices without breaking the gradient.

1. Decoding The first term asks that the generated code be easy to decode. We simulate the procedure of the decoding task. We sample a depolarizing error at physical rate p under the code-capacity noise model. Writing $e_X, e_Z \in \{0,1\}^n$ for the two components of that error, the syndromes are formed against the continuous check matrices through equation 15,

$$s_Z = \mathrm{par}\big(e_X H_Z^{\mathsf{T}}\big), \qquad s_X = \mathrm{par}\big(e_Z H_X^{\mathsf{T}}\big). \tag{16}$$

A BP decoder is then unrolled (Nachmani et al., 2016) for a fixed number of iterations, with damping and a constant channel log-likelihood ratio $\lambda_{\mathrm{ch}} = \log \frac{1-2p/3}{2p/3}$, producing soft estimates $\widehat{e}_X = \mathrm{BP}(H_Z, s_Z)$ and $\widehat{e}_Z = \mathrm{BP}(H_X, s_X)$. Unrolling is what keeps the decoder differentiable, every message update is an arithmetic operation on H_X and H_Z, so the error the decoder makes is backpropagated through the decoding dynamics, through the LP, and into the model's logits. The term is the decoder's residual uncertainty about the true error (see Appendix A):

$$E_{\mathrm{LER}} \;=\; \tfrac{1}{2}\Big[\mathrm{BCE}\big(\widehat{e}_X,\, e_X\big) + \mathrm{BCE}\big(\widehat{e}_Z,\, e_Z\big)\Big], \tag{17}$$

2. Geometry The next four terms act on the unfolded base matrix $\mathcal{M}$. We evaluated them on H_X and H_Z as well and no difference was found in the resulting codes, therefore we used $\mathcal{M}$ for more efficient compute. Let us define the following matrices:

$$A_c = \mathcal{M}\mathcal{M}^{\mathsf{T}} \in \mathbb{R}^{\bar{m}\times\bar{m}}, \qquad A_v = \mathcal{M}^{\mathsf{T}}\mathcal{M} \in \mathbb{R}^{\bar{n}\times\bar{n}} \tag{18}$$

for the two node-projection matrices, an off-diagonal entry of A_c counts the variable nodes shared by a pair of checks, and symmetrically for A_v. Short cycles can affect BP performance, so the second and third terms suppress them. A 4-cycle is a pair of nodes sharing two neighbours, which is exactly an off-diagonal entry of A_c or A_v exceeding one, so

$$E_{\mathrm{girth}_4} \;=\; \frac{\sum_{i\neq j}(A_c)_{ij}^2 \;+\; \sum_{i\neq j}(A_v)_{ij}^2}{\bar{m}(\bar{m}-1) + \bar{n}(\bar{n}-1)}, \tag{19}$$

normalized by the number of ordered off-diagonal pairs for a size-independent scale.

The third term suppresses 6-cycles. Writing $B_c = A_c \odot (\mathbf{1} - I)$ and $B_v = A_v \odot (\mathbf{1} - I)$ for the projections with diagonals removed, $\mathrm{tr}(B^3)$ counts closed triples, but over-counts. A single node of degree three or more creates triangles that are not bipartite 6-cycles. We therefore subtract those spurious contributions. We define: $S_r^v = \sum_i \mathcal{M}_{ij}^{2r}$ per column and S_r^c the analogous row quantity,

$$E_{\mathrm{girth}_6} = \frac{\big[\mathrm{tr}(B_c^3) - \Sigma_c\big]_+}{\bar{m}(\bar{m}-1)(\bar{m}-2)} + \frac{\big[\mathrm{tr}(B_v^3) - \Sigma_v\big]_+}{\bar{n}(\bar{n}-1)(\bar{n}-2)}, \quad \Sigma_\bullet = \sum\big(S_1^3 - 3S_1S_2 + 2S_3\big), \tag{20}$$

where $[\,\cdot\,]_+$ clamps at zero against floating-point drift and Σ_c uses the column sums, Σ_v the row sums. Driving both terms down pushes the Tanner girth to 8 or above.

The fourth term pushes down the second-largest eigenvalue of the two node projections, each normalized by its diagonal part (Tanner, 2001) $D_c = \mathrm{diag}(A_c)$, $D_v = \mathrm{diag}(A_v)$:

$$E_{\mathrm{spectral}} \;=\; \mu_2\Big(D_c^{-1/2} A_c D_c^{-1/2}\Big) \;+\; \mu_2\Big(D_v^{-1/2} A_v D_v^{-1/2}\Big)\,. \tag{21}$$

We use this as a heuristic for the code's distance, since it assumes a regular binary graph, whereas during sampling $\mathcal{M}$ is continuous.

The fifth term prevents a rank collapse. We use the log-determinant over the singular values s_i:

$$E_{\mathrm{rank}} \;=\; -\frac{1}{\min(\bar{m},\bar{n})}\sum_i \log s_i(\mathcal{M}), \tag{22}$$

which diverges as any singular value approaches zero, penalizing directions in which rank is lost.

3. Hardware The sixth term aims to keep the check weights light. We use a soft maximum of the row and column sums of $\mathcal{M}$:

$$E_{\mathrm{depth}} \;=\; \frac{1}{\beta}\log\sum_i e^{\beta r_i} \;+\; \frac{1}{\beta}\log\sum_j e^{\beta c_j}, \qquad r_i = \sum_j \mathcal{M}_{ij}, \quad c_j = \sum_i \mathcal{M}_{ij}, \tag{23}$$

The gradient is distributed across the rows in proportion to the exponential of their sums, so the heaviest rows dominate while every row retains a nonzero contribution. Increasing β sharpens the term toward the true maximum. The last two terms act directly on $\widehat{M}_0$ rather than its unfolding. The seventh term is exactly the degree term used in training 11, holding the support degrees at their targets during sampling. The eighth is the plain L_1 norm of the base matrix, the expected number of ones, to maintain a general sparse structure for the final output: $E_{L_1} \;=\; \sum_{i,j,k} \widehat{M}_{0,ijk}$,

Table 1: The strongest codes generated by the model. k/n is the encoding rate and w_{chk} the maximum check weight, the largest row weight over both H_X and H_Z.

#	$[[n,k,d]]$	k/n	w_{chk}	#	$[[n,k,d]]$	k/n	w_{chk}
1	$[[260,20,6]]$	0.077	5	17	$[[156,20,8]]$	0.128	8
2	$[[260,20,8]]$	0.077	6	18	**[[247, 19, 14]]**	0.077	8
3	$[[260,20,11]]$	0.077	7	19	$[[325,45,8]]$	0.138	9
4	$[[182,14,11]]$	0.077	7	20	$[[400,80,6]]$	0.200	9
5	$[[247,19,11]]$	0.077	7	21	$[[312,24,10]]$	0.077	9
6	**[[169, 13, 11]]**	0.077	7	22	$[[169,13,12]]$	0.077	9
7	$[[592,32,13]]$	0.054	7	23	$[[493,153,6]]$	0.310	9
8	$[[65,5,7]]$	0.077	7	24	**[[340, 68, 9]]**	0.200	9
9	$[[117,9,9]]$	0.077	7	25	$[[160,32,6]]$	0.200	9
10	$[[91,13,5]]$	0.143	7	26	$[[195,27,6]]$	0.138	10
11	$[[592,32,14]]$	0.054	8	27	$[[480,96,7]]$	0.200	10
12	$[[91,7,8]]$	0.077	8	28	$[[267,27,7]]$	0.101	10
13	**[[260, 52, 7]]**	0.200	8	29	**[[730, 180, 7]]**	0.247	11
14	**[[260, 20, 12]]**	0.077	8	30	$[[260,20,14]]$	0.077	11
15	$[[260,24,10]]$	0.092	8	31	$[[493,153,7]]$	0.310	11
16	$[[143,15,9]]$	0.105	8	32	$[[493,153,8]]$	0.310	12

The combined energy Finally, we obtain the combined energy:

$$\begin{aligned} E \;=\; & \lambda_{\text{LER}}\, E_{\text{LER}} + \lambda_{\text{girth}_4}\, E_{\text{girth}_4} + \lambda_{\text{girth}_6}\, E_{\text{girth}_6} + \lambda_{\text{spectral}}\, E_{\text{spectral}} \\ & + \lambda_{\text{rank}}\, E_{\text{rank}} + \lambda_{\text{depth}}\, E_{\text{depth}} + \lambda_{\text{deg}}\, E_{\text{deg}} + \lambda_{L_1}\, E_{L_1}, \end{aligned} \tag{24}$$

whose gradient is what equation 13 follows. We state the weights used for every reported run. After the reverse loop finishes, one further gradient step is taken directly on the continuous tensor to refine uncertain final probabilities. The tensor is then clipped to $[0,1]$ and rounded at $\frac{1}{2}$. This last step is scaled separately from g, and is specified per shape.

4.5 Screening and exact distance certification

Generation returns batches of candidate base matrices. Each generated sample is expanded into the corresponding parity-check matrices H_X and H_Z, which are first verified to satisfy the commutation condition $H_X H_Z^{\mathsf{T}} = 0$. For each valid code, compute its block length n and number of logical qubits $k = n - \operatorname{rank} H_X - \operatorname{rank} H_Z$. During evaluation, the distance of each generated code is estimated using the open-source upper-bound tool of Pryadko et al. (2022). The generated codes are subsequently ranked according to their n, k, distance upper bound and maximum check weight, where the latter is defined as the heaviest row weight across H_X and H_Z. As exact quantum-code distance computation is NP-hard in general (Kapshikar & Kundu, 2023), we use MILP to compute the exact distance of the final candidates, using the implementation of Cruz-Benito et al. (2026).

5 Results

5.1 Experimental setup

Our experimental evaluation follows the pipeline introduced in Section 4. We employ the network described in Section 4.2 as the denoiser in the diffusion process. The architecture consists of L layers, h attention heads, and an embedding dimension d. While the exact configuration varies per setting, the parameter count is approximately 10.7M, depending slightly on ℓ. Training samples are generated on the fly. Due to the lack of relevant data and to provide a more structured training distribution than uniformly random matrices, each sample is constructed in two steps. First, a binary $m \times n$ support is generated using a PEG procedure (Hu et al., 2005). Second, each nonzero entry in the support is assigned a random polynomial with weight $w \sim \text{Uniform}\{1, \ldots, w_{\max}\}$, with w positions selected uniformly. We treat $w_{\max}$ as a hyperparameter, with the goal of providing the model with a basic sparse structural prior. The forward corruption process spans $T = 100$ steps under a cosine schedule, with β_t ranging from 10^{-4} to 2×10^{-2}. We train the model for 750 epochs

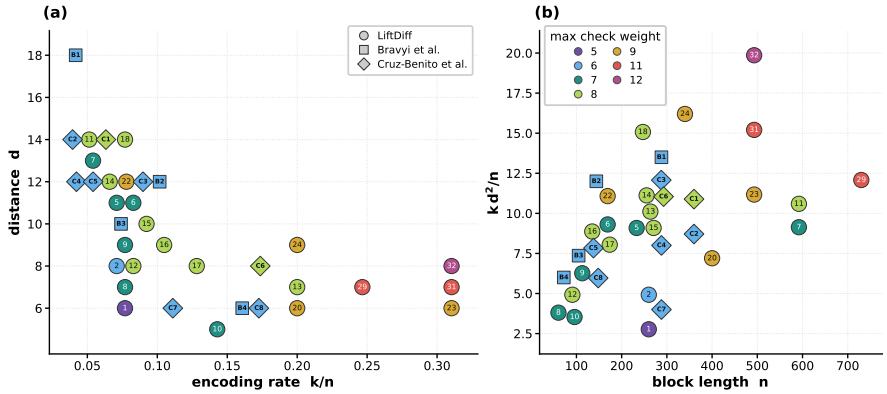


Figure 2: Selected codes from Table 1 against the baselines. Each marker is one code, colored by its w_{chk}. Circles are codes from this work, tagged with their row number in Table 1. Squares are BB codes of Bravyi et al. (2024) and diamonds the CSS codes of Cruz-Benito et al. (2026). **(a)** Encoding rate k/n against distance d. **(b)** Block length n against kd^2/n. Codes sharing a point exactly are offset by a small fraction of the axis. Baseline key: B1 $[[288, 12, 18]]$, B2 $[[144, 12, 12]]$, B3 $[[108, 8, 10]]$, B4 $[[72, 12, 6]]$; C1 $[[360, 20, \leq 14]]$, C2 $[[360, 16, \leq 14]]$, C3 $[[288, 24, 12]]$, C4 $[[288, 16, 12]]$, C5 $[[144, 8, 12]]$, C6 $[[288, 50, 8]]$, C7 $[[288, 32, 6]]$, C8 $[[144, 24, 6]]$.

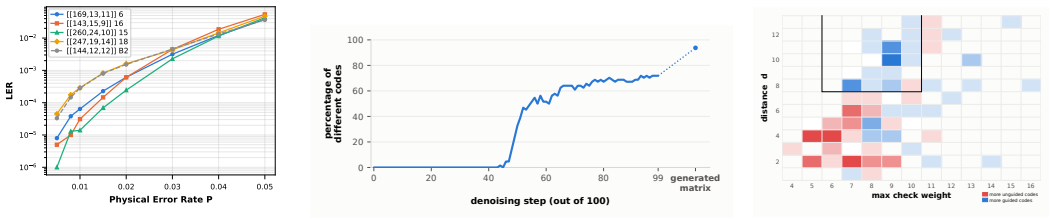
using AdamW (Loshchilov & Hutter, 2017) with a batch size of 16. In each epoch, 4096 base matrices are generated using the PEG sampler. The peak learning rate is 10^{-4} with a weight decay of 10^{-4}. The learning rate undergoes a linear warmup over the first 20 epochs before annealing to zero via cosine decay. Gradients are clipped to a maximum norm of 1.0, and the degree term in the loss function of equation 11 is weighted by $\lambda_{\text{deg}} = 3.0$. Training is performed on a single NVIDIA L40 GPU, with each epoch taking approximately 5 seconds. During inference, each run generates a batch of 64 samples using the guided reverse process defined in equation 13. Each run is initialized with a specific base geometry (m, n, ℓ), a target column weight d_v, and a maximum polynomial weight $w_{\max}$. The guidance scale and the hyperparameters of all energy terms are tuned individually for each run. Complete configurations for all generated results are provided in Appendix C.

5.2 Generated codes

We gathered 32 codes produced by LiftDiff, detailed in the accompanying Table 1. The discovered codes cover a broad range of $[[n, k, d]]$ parameters, with block lengths (n) ranging from 65 to 730, and distances (d) between 5 and 14. Our construction achieves high encoding rates (k/n), reaching up to 0.310 while keeping the maximum check weight (w_{chk}) within a range of 5 to 12. Several codes demonstrate a favorable balance between encoding rate, distance, and hardware-friendly check weights (for example 6, 13–16 and 18). To evaluate the performance of our construction, we benchmarked the new codes against the BB codes (Bravyi et al., 2024) and recent work (Cruz-Benito et al., 2026) in Figure 2. Panel (a) illustrates the trade-off between d and k/n. Our codes achieve high encoding rates, extending beyond 0.30. Code 18 occupies a favorable position, achieving high rate, $d = 14$ and $w_{chk} = 8$, compared to C1 and C2 which achieve lower rate and the same distance. Panel (b) evaluates the overall code efficiency kd^2/n as a function of the block length n. The results demonstrate that our method achieves high code efficiency across a range of block lengths. Notably, our codes reach kd^2/n values approaching 17 (code 24), significantly outperforming C1 and C2 while increasing w_{chk} by only one. Additionally, code 18 achieves the highest kd^2/n with $w_{chk} \leq 8$, at a modest block length within the range of the baseline codes.

5.3 Logical error rate

To evaluate the error-correction performance of the codes generated by LiftDiff, we measure the code-capacity logical error rate (LER) as a function of the physical error rate p as illustrated in Figure 3a. We consider independent X errors occurring on each qubit with probability p, and decode the resulting Z-syndrome using BP-OSD with product-sum BP for 20 iterations, followed by OSD with



(a) Code capacity LER (b) Code divergence over steps (c) Guided vs unguided codes

Figure 3: **(a)** Code-capacity block LER vs. physical error rate p for four LiftDiff codes and the [[144,12,12]] BB code. Each label's index identifies the code in Table 1. **(b)** Share of codes that differ from their unguided counterpart per denoising step. **(c)** Weight-distance cells where guidance produces more codes (blue) or fewer (red) than unguided generation. Distances are upper bounds.

CS order 7, following the simulation procedure of Cruz-Benito et al. (2026). For each value of p, we generate 10^6 independent error samples. Figure 3a shows the resulting LER curves for four codes generated by LiftDiff, together with the $[[144, 12, 12]]$ code for comparison. The $[[260, 24, 10]]$ and $[[169, 13, 11]]$ codes stay below the $[[144, 12, 12]]$ curve over almost the entire range, up to $p = 0.04$, and only meet the baseline at $p = 0.05$. The $[[143, 15, 9]]$ code is below it for $p \leq 0.02$, matches it at $p = 0.03$ and rises above it beyond that. The gap is widest at low physical error rates: at $p = 0.01$ the $[[260, 24, 10]]$ code reaches an LER about twenty times lower than the $[[144, 12, 12]]$ code, while $[[143, 15, 9]]$ and $[[169, 13, 11]]$ are lower by factors of about nine and four. The $[[247, 19, 14]]$ code behaves differently, and follows the $[[144, 12, 12]]$ curve closely across the whole range. The LER reported here is a block error rate, a sample is counted as a failure if any of the k logical qubits is flipped after decoding, rather than averaging over the individual logical qubits. This makes the result of the $[[260, 24, 10]]$ code notable, since it reaches the lowest curve while encoding twice as many logical qubits as the $[[144, 12, 12]]$ code. Appendix B provides circuit-level evaluation.

6 Model Analysis

To further evaluate the proposed method, we conducted additional experiments. In Figure 3b, we present two runs initialized with the same seed, one without the guidance mechanism and the other with guidance. The curve shows the percentage of differing matrices between the two batches throughout the denoising process. Since the two runs are otherwise identical, each matrix in one batch can be paired with the corresponding matrix in the other batch, allowing us to directly measure the effect of guidance on each sample. As the denoising process progresses, the percentage of differing matrices increases, such that by the end of the process, there are almost no identical matrices between the two batches. In Figure 3c, we examine how this divergence is reflected in the distance and maximum check weight of the resulting codes. The target region corresponds to high distance and a maximum check weight between 6 and 10, as indicated by the marked boundaries. This region is predominantly occupied by codes generated with the guidance mechanism, aggregated across multiple random seeds. In contrast, majority of codes generated by the model without guidance are concentrated at lower maximum check weights, but also exhibit lower distances. This separation indicates that, while the unguided model tends to generate codes with lower check weights, the guidance mechanism steers the generation process toward regions of the search space that are more relevant to our target criteria, allowing us to reach codes with higher distances while maintaining a maximum check weight in the desired range.

7 Conclusion

In this work, we introduced LiftDiff, the first discrete diffusion model designed for generating QECC, specifically, LP codes. Our framework demonstrates that generative models provide a highly effective strategy for navigating the vast search space of QECC. The experimental results yield robust codes characterized by competitive encoding rates, code distances, and low LER.

AI USE STATEMENT

During the making of this work, LLMs were used to assist with the following tasks: improving academic writing, primarily with respect to grammar, clarity, and style, as well as serving as an aid tool during code development and debugging. We verified all assisted work. The main development of the code, writing of the paper, mathematical formulations, methodological ideas, and contributions presented in this work were developed by the authors.

REPRODUCIBILITY STATEMENT

The experimental setup 5.1 provides detailed training configurations, the dataset used, and the fixed parameters. Appendix C provides the hyperparameters and settings used for each code presented in the paper, completing the details provided in the experimental setup. The code is included in the supplementary material and will be released as a public repository upon acceptance.

REFERENCES

Yuri Alexeev, Marwa H. Farag, Taylor L. Patti, Mark E. Wolf, Natalia Ares, Alán Aspuru-Guzik, Simon C. Benjamin, Zhenyu Cai, Shuxiang Cao, Christopher Chamberland, Zohim Chandani, Federico Fedele, Ikko Hamamura, Nicholas Harrigan, Jin-Sung Kim, Elica Kyoseva, Justin G. Lietz, Tom Lubowe, Alexander McCaskey, Roger G. Melko, Kouhei Nakaji, Alberto Peruzzo, Pooja Rao, Bruno Schmitt, Sam Stanwyck, Norm M. Tubman, Hanrui Wang, and Timothy Costa. Artificial intelligence for quantum computing. *Nature Communications*, 16(1), December 2025. ISSN 2041-1723. doi: 10.1038/s41467-025-65836-3. URL http://dx.doi.org/10.1038/s41467-025-65836-3.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Johannes Bausch, Andrew W Senior, Francisco JH Heras, Thomas Edlich, Alex Davies, Michael Newman, Cody Jones, Kevin Satzinger, Murphy Yuezhen Niu, Sam Blackwell, et al. Learning high-accuracy error decoding for quantum processors. *Nature*, 635(8040):834–840, 2024.

John Blue, Harshil Avlani, Zhiyang He, Liu Ziyin, and Isaac L. Chuang. Machine learning decoding of circuit-level noise for bivariate bicycle codes. *Quantum*, 10:2149, 2026. ISSN 2521-327X. doi: 10.22331/q-2026-06-30-2149. URL http://dx.doi.org/10.22331/q-2026-06-30-2149.

S. B. Bravyi and A. Yu. Kitaev. Quantum codes on a lattice with boundary, 1998. URL https://arxiv.org/abs/quant-ph/9811052.

Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, March 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07107-7. URL http://dx.doi.org/10.1038/s41586-024-07107-7.

Nikolas P. Breuckmann and Jens Niklas Eberhardt. Quantum low-density parity-check codes. *PRX Quantum*, 2(4), October 2021. ISSN 2691-3399. doi: 10.1103/prxquantum.2.040101. URL http://dx.doi.org/10.1103/PRXQuantum.2.040101.

Nikolas P. Breuckmann and Barbara M. Terhal. Constructions and noise threshold of hyperbolic surface codes. *IEEE Transactions on Information Theory*, 62(6):3731–3744, 2016. ISSN 1557-9654. doi: 10.1109/tit.2016.2555700. URL http://dx.doi.org/10.1109/TIT.2016.2555700.

A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, Aug 1996. doi: 10.1103/PhysRevA.54.1098. URL https://link.aps.org/doi/10.1103/PhysRevA.54.1098.

Christopher Chamberland, Jan Olle, Muyuan Li, Scott Thornton, and Igor Baratta. Fast and accurate ai-based pre-decoders for surface codes. *arXiv preprint arXiv:2604.12841*, 2026.

Yihua Chengyu, Richard Meister, Conor Carty, Sheng-Ku Lin, and Roberto Bondesan. Bayesian optimization for quantum error-correcting code discovery, 2026. URL https://arxiv.org/abs/2601.18562.

Yoni Choukroun and Lior Wolf. Deep quantum error correction, 2023. URL https://arxiv.org/abs/2301.11930.

Juan Cruz-Benito, Andrew W. Cross, David Kremer, and Ismael Faro. Evolutionary discovery of bivariate bicycle codes with llm-guided search, 2026. URL https://arxiv.org/abs/2606.02418.

Nicolas Delfosse and Naomi H. Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, December 2021. ISSN 2521-327X. doi: 10.22331/q-2021-12-02-595. URL http://dx.doi.org/10.22331/q-2021-12-02-595.

Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002. ISSN 1089-7658. doi: 10.1063/1.1499754. URL http://dx.doi.org/10.1063/1.1499754.

Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

Richard P Feynman et al. Simulating physics with computers. *Feynman and computation*, 22: 133–153, 2018.

Austin G. Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time, 2014. URL https://arxiv.org/abs/1307.1740.

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

Daniel Gottesman. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

Andi Gu, J. Pablo Bonilla Ataides, Mikhail D. Lukin, and Susanne F. Yelin. Scalable neural decoders for practical fault-tolerant quantum computation, 2026. URL https://arxiv.org/abs/2604.08358.

Xiao-Yu Hu, E. Eleftheriou, and D.M. Arnold. Regular and irregular progressive edge-growth tanner graphs. *IEEE Transactions on Information Theory*, 51(1):386–398, 2005. doi: 10.1109/TIT.2004.839541.

Upendra Kapshikar and Srijita Kundu. On the hardness of the minimum distance problem of quantum codes. *IEEE Transactions on Information Theory*, 69(10):6293–6302, October 2023. ISSN 1557-9654. doi: 10.1109/tit.2023.3286870. URL http://dx.doi.org/10.1109/TIT.2023.3286870.

Kenta Kasai. Breaking the orthogonality barrier in quantum ldpc codes. *Quantum*, 10:2205, 2026. ISSN 2521-327X. doi: 10.22331/q-2026-09-09-2205. URL http://dx.doi.org/10.22331/q-2026-09-09-2205.

A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, January 2003. ISSN 0003-4916. doi: 10.1016/s0003-4916(02)00018-0. URL http://dx.doi.org/10.1016/S0003-4916(02)00018-0.

Mario Krenn, Jonas Landgraf, Thomas Foesel, and Florian Marquardt. Artificial intelligence and machine learning for quantum technologies. *Physical Review A*, 107(1), January 2023. ISSN 2469-9934. doi: 10.1103/physreva.107.010101. URL http://dx.doi.org/10.1103/PhysRevA.107.010101.

Hee-Youl Kwak, Seong-Joon Park, Dae-Young Yun, Eliya Nachmani, and Jae-Won Kim. Logical neural belief propagation for linear-complexity decoding of surface codes, 2026. URL https://arxiv.org/abs/2608.27682.

Moritz Lange, Pontus Havström, Basudha Srivastava, Isak Bengtsson, Valdemar Bergentall, Karl Hammar, Olivia Heuts, Evert van Nieuwenburg, and Mats Granath. Data-driven decoding of quantum error correcting codes using graph neural networks. *Physical Review Research*, 7(2), May 2025. ISSN 2643-1564. doi: 10.1103/physrevresearch.7.023181. URL http://dx.doi.org/10.1103/PhysRevResearch.7.023181.

Zidu Liu and Florian Marquardt. Large-language-model discovery of quantum ldpc codes through structured concept evolution, 2026. URL https://arxiv.org/abs/2606.24808.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Tristan Müller, Thomas Alexander, Michael E. Beverland, Markus Bühler, Blake R. Johnson, Thilo Maurer, and Drew Vandeth. Improved belief propagation is sufficient for real-time decoding of quantum memory, 2025. URL https://arxiv.org/abs/2506.01779.

Eliya Nachmani, Yair Be'Ery, and David Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346. IEEE, 2016.

Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J. Briegel, and Nicolai Friis. Optimizing quantum error correction codes with reinforcement learning. *Quantum*, 3:215, December 2019. ISSN 2521-327X. doi: 10.22331/q-2019-12-16-215. URL http://dx.doi.org/10.22331/q-2019-12-16-215.

Jan Olle, Remmy Zen, Matteo Puviani, and Florian Marquardt. Simultaneous discovery of quantum error correction codes and encoders with a noise-aware reinforcement learning agent. *npj Quantum Information*, 10(1):126, 2024.

Pavel Panteleev and Gleb Kalachev. Degenerate quantum ldpc codes with good finite length performance. *Quantum*, 5:585, November 2021. ISSN 2521-327X. doi: 10.22331/q-2021-11-22-585. URL http://dx.doi.org/10.22331/q-2021-11-22-585.

Pavel Panteleev and Gleb Kalachev. Quantum ldpc codes with almost linear minimum distance. *IEEE Transactions on Information Theory*, 68(1):213–229, January 2022. ISSN 1557-9654. doi: 10.1109/tit.2021.3119384. URL http://dx.doi.org/10.1109/TIT.2021.3119384.

Yotam Peled, David Zenati, and Eliya Nachmani. Neural minimum weight perfect matching for quantum error codes, 2026. URL https://arxiv.org/abs/2601.00242.

Michael A. Perlin. qLDPC. https://github.com/qLDPCOrg/qLDPC, 2023.

Leonid P. Pryadko, Vadim A. Shabashov, and Valerii K. Kozin. Qdistrnd: A gap package for computing the distance of quantum error-correcting codes. *Journal of Open Source Software*, 7(71): 4120, March 2022. ISSN 2475-9066. doi: 10.21105/joss.04120. URL http://dx.doi.org/10.21105/joss.04120.

Joschka Roffe. Quantum error correction: an introductory guide. *Contemporary Physics*, 60(3): 226–245, 2019. ISSN 1366-5812. doi: 10.1080/00107514.2019.1667078. URL http://dx.doi.org/10.1080/00107514.2019.1667078.

Omer S. Sella, Robert Pinsler, and Thomas Heinis. Designing quantum error correcting codes to fit decoders via reinforcement learning, 2026. URL https://arxiv.org/abs/2608.15754.

Andrew W. Senior, Thomas Edlich, Francisco J. H. Heras, Lei M. Zhang, Oscar Higgott, James S. Spencer, Taylor Applebaum, Sam Blackwell, Justin Ledford, Akvilė Žemgulytė, Augustin Žídek, Noah Shutty, Andrew Cowie, Yin Li, George Holland, Peter Brooks, Charlie Beattie, Michael Newman, Alex Davies, Cody Jones, Sergio Boixo, Hartmut Neven, Pushmeet Kohli, and Johannes Bausch. A scalable and real-time neural decoder for topological quantum codes, 2026. URL https://arxiv.org/abs/2512.07737.

Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52: R2493(R)–R2496(R), Oct 1995. doi: 10.1103/PhysRevA.52.R2493. URL https://link.aps.org/doi/10.1103/PhysRevA.52.R2493.

Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997. ISSN 1095-7111. doi: 10.1137/s0097539795293172. URL http://dx.doi.org/10.1137/S0097539795293172.

Andrew M Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793, 1996.

Vincent Paul Su, ChunJun Cao, Hong-Ye Hu, Yariv Yanay, Charles Tahan, and Brian Swingle. Discovery of optimal quantum codes via reinforcement learning. *Phys. Rev. Appl.*, 23:034048, Mar 2025. doi: 10.1103/PhysRevApplied.23.034048. URL https://link.aps.org/doi/10.1103/PhysRevApplied.23.034048.

R.M. Tanner. Minimum-distance bounds by graph analysis. *IEEE Transactions on Information Theory*, 47(2):808–821, 2001. doi: 10.1109/18.910591.

Jean-Pierre Tillich and Gilles Zemor. Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60 (2):1193–1202, February 2014. ISSN 1557-9654. doi: 10.1109/tit.2013.2292061. URL http://dx.doi.org/10.1109/TIT.2013.2292061.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zihao Wang and Hao Tang. Artificial intelligence for quantum error correction: A comprehensive review, 2024. URL https://arxiv.org/abs/2412.20380.

Yue Wu, Namitha Liyanage, and Lin Zhong. Micro blossom: Accelerated minimum-weight perfect matching decoding for quantum error correction. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 639–654. ACM, March 2025. doi: 10.1145/3676641.3716005. URL http://dx.doi.org/10.1145/3676641.3716005.

Ge Yan, Shanchuan Li, and Yuxuan Du. Rethink the role of neural decoders in quantum error correction, 2026a. URL https://arxiv.org/abs/2605.12046.

Ge Yan, Shanchuan Li, Pengyue Ma, Qixin Zhang, Pingchuan Ma, Jianping Wang, Min-Hsiu Hsieh, and Yuxuan Du. Omniqec: discovering practical quantum error-correcting codes by an ai scientist, 2026b. URL https://arxiv.org/abs/2607.25865.

David Zenati and Eliya Nachmani. Saq: Stabilizer-aware quantum error correction decoder, 2026. URL https://arxiv.org/abs/2512.08914.

Chen Zhao, Casey Duckering, Andi Gu, Nishad Maskara, and Hengyun Zhou. Towards ultra-high-rate quantum error correction with reconfigurable atom arrays, 2026. URL https://arxiv.org/abs/2604.16209.

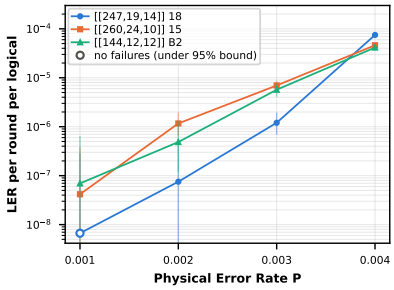


Figure 4: Circuit-level LER per logical qubit per round vs. physical error rate p for two codes produced by LiftDiff and the $[[144, 12, 12]]$ BB code. Number of rounds equal to each code's distance. The open marker denotes a point with no observed failures, showing the 95% upper bound. The index beside each label identifies the code in Table 1.

A Belief-propagation configuration

E_{LER} uses sum–product BP, unrolled for 20 iterations and run once per error type (H_Z for e_X, H_X for e_Z). All messages are recomputed in parallel at each iteration, and the loop always runs to its full depth rather than stopping once the syndrome is matched. Variable-to-check messages are damped: each is 0.8 of its newly computed value plus 0.2 of its previous one. Every edge is weighted by the corresponding entry of H_X or H_Z, so an entry of 1 gives the ordinary update and an entry of 0 removes the edge, which is what lets the decoder run on the relaxed check matrices. The physical error rate is $p = 0.05$, so the channel log-likelihood ratio is 3.37 on every variable node, and each evaluation of E_{LER} averages 64 sampled error patterns. The arguments of $\tanh$ and artanh are clamped to ± 0.9999 and the posterior to $[10^{-6}, 1 - 10^{-6}]$. All of these are fixed for every run.

B Circuit-level evaluation

To evaluate the performance of the generated codes under circuit-level noise, we measure the LER per logical qubit per round as a function of the physical error rate p. The experiment covers the $[[247, 19, 14]]$ and $[[260, 24, 10]]$ codes of Table 1, together with the $[[144, 12, 12]]$ code as a baseline. Every code is compiled into a syndrome-extraction circuit and its detector error model under the same protocol (Perlin, 2023), to provide an equal comparison. We repeat the syndrome measurement for a number of rounds equal to the code distance, 14, 10 and 12 respectively, and measure once in the X basis and once in the Z basis. We decode with the relay BP decoder (Müller et al., 2025) with a stop after the first converged solution, under circuit-level depolarizing noise of strength p for $p \in \{0.001, 0.002, 0.003, 0.004\}$. Each point draws up to 10^5 shots per basis and stops once it has recorded 200 failures. Figure 4 shows the resulting curves. The $[[247, 19, 14]]$ code stays below the $[[144, 12, 12]]$ curve from $p = 0.001$ to $p = 0.003$. It records no failures at $p = 0.001$, At $p = 0.004$ the baseline takes a small margin. The $[[260, 24, 10]]$ code follows the $[[144, 12, 12]]$ curve closely over the whole range and stays within its confidence intervals at every point.

C Detailed Codes Configuration

Table 2: Every code of Table 1 together with the settings of the run that produced it. $m \times n \times \ell$ is the base geometry, d_v the target column weight and $w_{\max}$ the largest polynomial weight the training data was built with. g is the guidance scale of equation 13 and g_{fin} the scale of the single gradient step taken on the continuous tensor before binarization. $= g$ marks configuration where the same scale was used in both. The remaining columns are the energy weights of equation 24. Every other setting was the same in all experiments: 100 diffusion steps on a cosine β schedule from 10^{-4} to 2×10^{-2}, a 256-wide transformer with 4 heads and 6 layers, except where a row is marked. 750 epochs of 4096 base matrices produced by PEG, in batches of 16, optimized by AdamW at learning rate 10^{-4} with weight decay 10^{-4}, 20 warm-up epochs, cosine annealing and gradient clipping at 1, a training-time degree weight of 3 with margin 0, and 64 samples drawn.

#	$[[n,k,d]]$	$m \times n \times \ell$	d_v	$w_{\max}$	g	g_{fin}	λ_{LER}	λ_{girth_4}	λ_{girth_6}	$\lambda_{\text{spectral}}$	λ_{rank}	λ_{depth}	λ_{deg}	λ_{L_1}
1	$[[260,20,6]]$	$4 \times 6 \times 5$	2	1	2	$= g$	15	0.15	0.015	0.4	0.2	0.1	0.25	0.025
2	$[[260,20,8]]$	$4 \times 6 \times 5$	2	1	2	$= g$	1	0.15	0.015	0.25	0.5	0.1	0.25	0.02
3[ab]	$[[260,20,11]]$	$2 \times 3 \times 20$	2	3	2	$= g$	10	0.15	0.01	0.5	0.5	0.1	0.25	0.01
4	$[[182,14,11]]$	$2 \times 3 \times 14$	2	2	1	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.02
5	$[[247,19,11]]$	$2 \times 3 \times 19$	2	2	2	0.5	10	0.15	0.015	0.6	0.5	0.1	0.3	0.02
6	$[[169,13,11]]$	$2 \times 3 \times 13$	2	3	1	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.02
7	$[[592,32,13]]$	$5 \times 7 \times 8$	3	1	2	$= g$	15	0.15	0.015	0.5	0.3	0.1	0.25	0.01
8	$[[65,5,7]]$	$2 \times 3 \times 5$	2	1	3	$= g$	10	0.15	0.01	0.5	0.5	0.1	0.25	0.03
9	$[[117,9,9]]$	$2 \times 3 \times 9$	2	2	2	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.02
10	$[[91,13,5]]$	$2 \times 3 \times 7$	2	2	1	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.04
11	$[[592,32,14]]$	$5 \times 7 \times 8$	3	1	2	$= g$	15	0.15	0.015	0.5	0.3	0.1	0.25	0.025
12	$[[91,7,8]]$	$2 \times 3 \times 7$	2	2	1	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.04
13	$[[260,52,7]]$	$2 \times 4 \times 13$	2	2	2	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.03
14[ab]	$[[260,20,12]]$	$2 \times 3 \times 20$	2	2	1	$= g$	10	0.15	0.01	0.5	0.5	0.1	0.25	0.01
15	$[[260,24,10]]$	$2 \times 3 \times 20$	2	3	1	0.5	10	0.15	0.01	0.7	0.5	0.1	0.25	0.03
16[c]	$[[143,15,9]]$	$2 \times 3 \times 11$	2	1	1	0.5	10	0.15	0.015	0.5	0.2	0.1	0.2	0.01
17	$[[156,20,8]]$	$2 \times 3 \times 12$	2	2	1	0.5	10	0.15	0.015	0.5	0.2	0.1	0.2	0.04
18	$[[247,19,14]]$	$2 \times 3 \times 19$	2	2	5	0.5	10	0.15	0.015	0.55	0.5	0.1	0.3	0.03
19	$[[325,45,8]]$	$4 \times 7 \times 5$	3	1	2	$= g$	10	0.15	0.015	0.5	0.5	0.1	0.25	0.02
20	$[[400,80,6]]$	$4 \times 8 \times 5$	3	1	2	$= g$	20	0.15	0.015	0.6	0.3	0.11	0.25	0.025
21	$[[312,24,10]]$	$4 \times 6 \times 6$	3	1	2	$= g$	10	0.15	0.015	0.5	0.5	0.1	0.25	0.02
22	$[[169,13,12]]$	$2 \times 3 \times 13$	2	3	1	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.02
23	$[[493,153,6]]$	$2 \times 5 \times 17$	2	2	1	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.02
24	$[[340,68,9]]$	$2 \times 4 \times 17$	2	2	2	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.03
25	$[[160,32,6]]$	$2 \times 4 \times 8$	2	2	2	0.5	10	0.15	0.01	0.5	0.5	0.1	0.25	0.02
26	$[[195,27,6]]$	$4 \times 7 \times 3$	3	1	2	$= g$	1	0.1	0	0.5	0.5	0.1	0.25	0
27	$[[480,96,7]]$	$4 \times 8 \times 6$	3	1	2	$= g$	1	0.15	0.015	0.25	0.5	0.1	0.25	0.02
28	$[[267,27,7]]$	$5 \times 8 \times 3$	3	1	2	$= g$	1	0.15	0.015	0.25	0.5	0.1	0.25	0.02
29	$[[730,180,7]]$	$5 \times 11 \times 5$	3	1	2	$= g$	15	0.15	0.015	0.5	0.3	0.1	0.25	0.02
30	$[[260,20,14]]$	$2 \times 3 \times 20$	2	2	2	0.5	10	0.15	0.015	0.6	0.5	0.1	0.3	0.04
31	$[[493,153,7]]$	$2 \times 5 \times 17$	2	2	2	0.5	10	0.15	0.01	0.7	0.5	0.1	0.25	0.04
32	$[[493,153,8]]$	$2 \times 5 \times 17$	2	2	1	0.5	10	0.15	0.01	0.7	0.5	0.1	0.25	0.04

[a] 1 attention head. [b] 3 layers. [c] learning rate 9×10^{-5}.