ZEROCODE: ON-DEMAND ERROR-CORRECTING CODE CONSTRUCTION FROM THE ZERO MATRIX VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Error-correcting codes (ECCs) are essential across diverse applications—from wireless communications and storage to quantum computing—yet each application imposes distinct design requirements on the parity-check matrix (PCM). To address these on-demand requirements in a unified framework, we propose *ZeroCode*[1], a reinforcement learning (RL)-based approach that constructs PCMs sequentially from the all-zero matrix. ZeroCode formulates construction as a discrete sequential decision-making problem and uses proximal policy optimization with action masking to select valid edges. ZeroCode achieves a gain of approximately 1 dB over the prior RL-based construction method at a bit error rate (BER) of 10^{-4} for the (32,16) code and outperforms existing genetic, differentiable, and classical code-design methods in our experiments. Beyond optimizing decoding performance, the masking mechanism allows on-demand structural constraints—such as a maximum degree, 4-cycle-free structure, and quasi-cyclic structure—to be flexibly incorporated. Moreover, a single policy rollout yields a library of PCMs with varying edge counts, offering trade-offs between decoding performance and complexity without retraining. Overall, ZeroCode addresses diverse code-design requirements within a unified framework, providing solutions with optimized decoding performance under given constraints.

1 INTRODUCTION

Error-correcting codes (ECCs) are a foundational technology for ensuring reliability, and their applications continue to expand across diverse domains including communication systems (Richardson & Kudekar, 2018), memory devices (Zhao et al., 2013), distributed storage (Ramkumar et al., 2022), and quantum computing (Shor, 1995). While decoding performance is universally important, each application imposes distinct design requirements on the parity-check matrix (PCM), such as complexity constraints, efficient encodability, and rate compatibility. No single hand-crafted code family can simultaneously satisfy this diverse set of requirements, motivating a flexible, on-demand approach to PCM design.

Early design for low-density parity-check (LDPC) codes primarily focused on asymptotic performance (Gallager, 1962; MacKay, 1999; Richardson et al., 2001): Degree distributions were optimized as proxies for PCMs to avoid a combinatorial search over individual PCMs while maximizing performance in the infinite-blocklength regime. While effective for long block lengths, this approach does not directly optimize the metric most relevant in practice—the bit error rate (BER) of a finite-length code. More recent work has addressed finite-length code design by optimizing the PCM itself, using genetic algorithms (GA) (Elkelesh et al., 2019), differentiable belief propagation (BP) (Choukroun & Wolf, 2024a), or reinforcement learning (RL) (Tian et al., 2025), with error performance as the optimization objective. Such finite-length optimization is particularly well-suited to short block lengths for low-latency communications (Yue et al., 2023; Shirvanimoghaddam et al., 2019), where asymptotic metrics are less informative.

Among these approaches, the RL-based approach of Tian et al. (2025) is most closely related to our work, yet it has several fundamental limitations. It employs deep deterministic policy gradient

[1]Code is available at `https://anonymous.4open.science/r/ppo_code-8D3D`.

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.

(DDPG) (Lillicrap et al., 2016), an algorithm designed for continuous action spaces, whereas PCM construction is inherently discrete—each edge is either included or excluded. This mismatch can result in inaccurate gradient estimates (Chen et al., 2019). Furthermore, the method initializes from a non-zero matrix and updates multiple edges at each step, making optimization sensitive to initialization and the search trajectory difficult to control.

In this paper, we address these limitations with *ZeroCode*, an RL framework that constructs PCMs sequentially by adding one edge per step from the all-zero matrix. This formulation removes dependence on the choice of a pre-constructed initial PCM and enables stable learning. It also naturally induces a Markov decision process (MDP) formulation that facilitates the incorporation of design-specific constraints via masking. We employ proximal policy optimization (PPO) to learn the policy for this discrete sequential construction problem. Experimental results show that ZeroCode significantly outperforms the existing RL-based construction (Tian et al., 2025), establishing a new state of the art among RL-based approaches to code design. Compared with non-RL approaches (Richardson et al., 2001; Elkelesh et al., 2019; Choukroun & Wolf, 2024a), ZeroCode also achieves superior BER performance at short block lengths.

Beyond optimizing performance, ZeroCode incorporates *on-demand* structural constraints directly into the construction process via *action masking*. Moreover, a single rollout of the trained policy generates a library of codes spanning a wide range of edge counts, yielding multiple performance–complexity trade-off points without retraining. In contrast, prior approaches (Elkelesh et al., 2019; Choukroun & Wolf, 2024a) focus on optimizing a target PCM rather than explicitly constructing a library indexed by edge count.

The main contributions of this paper are as follows.

1. We formulate PCM construction as a sequential decision-making problem over a discrete action space and learn an edge-by-edge construction policy starting from the all-zero matrix.
2. ZeroCode significantly outperforms the existing RL-based method and achieves superior BER performance over non-RL approaches at block lengths $n \in \{64, 128, 256\}$.
3. Structural constraints such as 4-cycle-free graphs, quasi-cyclic structure, maximum degree, and rate compatibility can be flexibly incorporated via masking, enabling on-demand code design across diverse application requirements.
4. A single trained policy yields a library of PCMs spanning a wide range of edge counts along its rollout trajectory, providing favorable performance–complexity trade-offs across operating points.

2 Related work

Richardson et al. (2001) optimized degree distributions for asymptotic performance, while the progressive edge-growth (PEG) algorithm (Hu et al., 2005) constructs finite-length Tanner graphs. More recent methods optimize finite-length codes for a specified decoder: Elkelesh et al. (2019) used block error rate as the fitness function of a GA, while Choukroun & Wolf (2024a) optimized the PCM through differentiable BP. However, those works do not explicitly address the range of on-demand structural constraints considered in ZeroCode. The RL-based approach of Tian et al. (2025) formulates PCM construction as an MDP and learns a DDPG-based policy that modifies multiple edges per step from a pre-constructed PCM. ZeroCode differs in three respects: (i) it constructs PCMs edge by edge from the all-zero matrix; (ii) it employs PPO, which is better suited to the discrete action space; and (iii) it uses action masking to enforce structural constraints. Huang et al. (2020) also applied RL and GA to the construction of polar codes and other linear block codes, but did not consider enforcing PCM-specific structural constraints during the construction process.

A complementary line of research applies deep learning and RL to improve the decoding algorithm, including neural BP decoders (Nachmani et al., 2018; Kwak et al., 2023; 2025), transformer-based decoders (Choukroun & Wolf, 2022; 2024b; Park et al., 2025a;b), and RL-based decoding schedules (Habib et al., 2020; 2023). These methods are orthogonal to ours and can be combined with ZeroCode.

Beyond ECC, RL has proven effective in complex sequential decision-making over discrete spaces, with notable successes including game playing (Silver et al., 2016; OpenAI et al., 2019) and chip

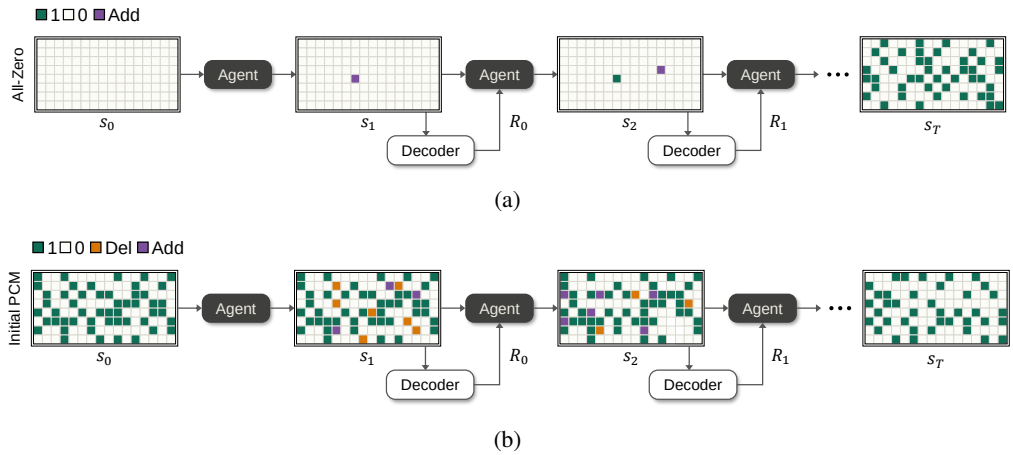


Figure 1: Overview of (a) ZeroCode compared with (b) the prior RL-based approach, DDPG (Tian et al., 2025). DDPG starts from a pre-constructed PCM and updates multiple edges at each step, whereas ZeroCode begins with an all-zero matrix and the agent incrementally adds a single edge per step. In both methods, the PCM at each step serves as the state, and the BER obtained from the decoder is used as the reward signal.

placement (Mirhoseini et al., 2021). The closest analogy is AlphaChip (Mirhoseini et al., 2021), which constructs chip layouts incrementally from an initially empty canvas, demonstrating the effectiveness of sequential construction for stable optimization and constraint-aware design. ZeroCode adopts the same incremental construction paradigm for PCM design.

3 Background

3.1 Error-correcting codes and parity-check matrix

An (n, k) code encodes an information message of length k into a codeword of length n to improve reliability against channel noise. A PCM $\mathbf{H} \in \{0,1\}^{m\times n}$ defines the code, where each row represents a parity-check constraint and each column corresponds to a coded bit. The set of valid codewords is the null space of $\mathbf{H}$ over $\mathrm{GF}(2)$. When $\mathbf{H}$ has full rank m, the information length is $k = n - m$ and the code rate is $r = k/n$. The goal of this work is to construct PCMs that yield strong decoding performance while satisfying application-specific structural constraints.

3.2 Decoding and performance evaluation

Given a PCM $\mathbf{H}$, a decoder estimates the transmitted codeword from the received noisy channel output, and its performance serves as the reward signal for our RL framework. While any decoder can in principle be used, we adopt BP as a representative example. BP iteratively exchanges LLR messages over the Tanner graph of $\mathbf{H}$ consisting of n variable nodes (VNs) and m check nodes (CNs). The message updates at the ℓ-th iteration are given by

$$\hat{m}_{c\to v}^{(\ell)} = 2\tanh^{-1}\left(\prod_{v'\in\mathcal{N}(c)\setminus\{v\}} \tanh\left(\frac{m_{v'\to c}^{(\ell-1)}}{2}\right)\right), \quad m_{v\to c}^{(\ell)} = m_v^{\mathrm{ch}} + \sum_{c'\in\mathcal{N}(v)\setminus\{c\}} \hat{m}_{c'\to v}^{(\ell)}, \quad (1)$$

where $\mathcal{N}(x)$ denotes the set of neighbor nodes of node x and m_v^{ch} is the channel LLR at VN v. At each iteration, the output LLR at VN v is $\bar{m}_v^{(\ell)} = m_v^{\mathrm{ch}} + \sum_{c\in\mathcal{N}(v)} \hat{m}_{c\to v}^{(\ell)}$, and the hard decision is 0 if $\bar{m}_v^{(\ell)} > 0$ and 1 otherwise. Decoding terminates when all parity checks are satisfied or the maximum number of iterations is reached. We evaluate each PCM in terms of the BER by the BP decoder.

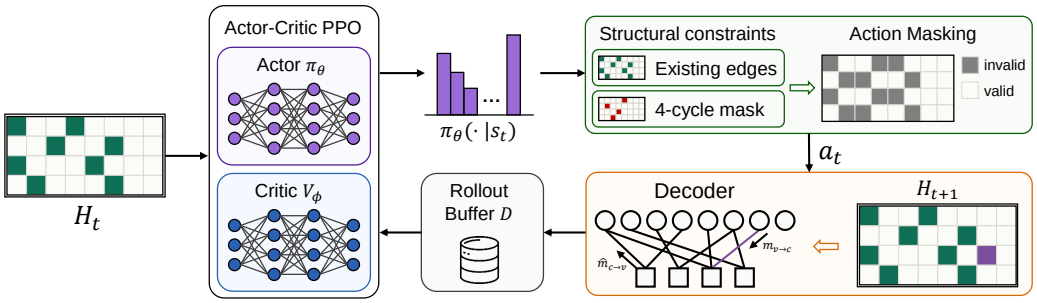


Figure 2: Overview of the PCM construction process in ZeroCode. The agent receives the current PCM as the state, applies an action mask to retain only edge candidates that satisfy the structural constraints, and samples a single edge from the masked policy. The updated matrix is evaluated by the BP decoder to produce the reward signal. Starting from the all-zero matrix, ZeroCode repeats this process to optimize PCMs.

3.3 Proximal Policy Optimization with Action Masking

RL models sequential decision-making as an MDP defined over a state space $\mathcal{S}$, an action space $\mathcal{A}$, and a reward function R. At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to its policy $\pi_\theta(\cdot \mid s_t)$ parameterized by θ, transitions to the next state s_{t+1}, and receives a reward R_t. The goal of RL is to learn a policy that maximizes the expected cumulative return.

We adopt PPO (Schulman et al., 2017) to learn the PCM construction policy. PPO is an actor–critic algorithm that jointly learns a policy $\pi_\theta(\cdot \mid s_t)$ and a value function $V_\phi(s_t)$. At each time step, an action is sampled as $a_t \sim \pi_\theta(\cdot \mid s_t)$, and the resulting transition $\tau_t := (s_t, a_t, R_t, s_{t+1})$ is stored in a rollout buffer $\mathcal{D}$. PPO updates (θ, ϕ) using samples from $\mathcal{D}$ by minimizing a combined loss $L^{\mathrm{PPO}}(\theta, \phi)$ consisting of a clipped policy loss, a value loss, and an entropy bonus. We refer to Schulman et al. (2017) for the precise definition of each term.

In our PCM construction problem, the valid action set varies with the current state: previously selected edges cannot be selected again, and edges that violate application-specific structural constraints are also excluded. We therefore combine PPO with action masking so that actions disallowed in the current state are never sampled. Let $\mathcal{A}_{\mathrm{valid}}(s_t) \subseteq \mathcal{A}$ denote the valid action set at state s_t. The masked policy $\tilde{\pi}_\theta(a \mid s_t)$ is obtained by renormalizing $\pi_\theta(a \mid s_t)$ over $\mathcal{A}_{\mathrm{valid}}(s_t)$:

$$\tilde{\pi}_\theta(a \mid s_t) = \frac{\pi_\theta(a \mid s_t)\,\mathbf{1}[a \in \mathcal{A}_{\mathrm{valid}}(s_t)]}{\sum_{a' \in \mathcal{A}} \pi_\theta(a' \mid s_t)\,\mathbf{1}[a' \in \mathcal{A}_{\mathrm{valid}}(s_t)]}. \tag{2}$$

As shown by Huang & Ontañón (2022), when the mask is independent of the policy parameter θ, the resulting policy-gradient estimator is unbiased for the gradient of the masked policy. Therefore, PPO can be applied with $\tilde{\pi}_\theta$ in place of π_θ without modifying the objective. This compatibility allows ZeroCode to incorporate structural constraints directly through masking while retaining the standard PPO training procedure.

4 Proposed Method

We formulate the construction of a binary PCM $\mathbf{H} \in \{0,1\}^{m \times n}$ as an RL problem. Starting from the all-zero matrix, the agent incrementally constructs $\mathbf{H}$ by selecting one edge at a time and setting it to 1, with the objective of optimizing decoding performance throughout the construction process. We define a PCM as valid when it has full rank over $\mathrm{GF}(2)$ and contains neither an all-zero row nor an all-zero column.

4.1 MDP Definition for ZeroCode

In this subsection, we define the state, action, and reward for PCM construction. An episode begins from the all-zero matrix and consists of a sequence of edge additions performed by the agent until

the termination condition is satisfied. Each individual edge addition within an episode corresponds to one step. The state s_t at step t is the current binary PCM, $s_t = \mathbf{H}_t \in \{0,1\}^{m\times n}$, with the initial state set to the all-zero matrix $\mathbf{H}_0 = \mathbf{0}_{m\times n}$. At each step, the agent selects a position in the matrix and adds an edge there. The episode terminates when no valid edge remains in the masked action set.

Every valid PCM encountered along the construction trajectory is retained in the code library. Because one edge is added at each step, the retained PCMs span different edge counts and, consequently, different decoding complexities. The resulting library allows an appropriate PCM to be selected according to application requirements.

After each step, BP decoding is run on the updated matrix $\mathbf{H}_{t+1}$ to compute BER_{t+1}. We define $\Lambda(\mathbf{H}_{t+1}) = -\ln \mathrm{BER}_{t+1}$ and use it as the reward:

$$R_t = R(s_t, a_t) = \Lambda(\mathbf{H}_{t+1}). \tag{3}$$

4.2 Action masking

In ZeroCode, action masking is used to restrict the valid action set at each step. We construct an action mask matrix $\mathbf{M}(\mathbf{H}_t) \in \{0,1\}^{m\times n}$, where $\mathbf{M}(\mathbf{H}_t)_{i,j} = 1$ indicates that edge (i,j) cannot be selected. By default, $\mathbf{M}(\mathbf{H}_t) = \mathbf{H}_t$, since edges, once added, cannot be removed. Additional positions can be masked to enforce structural constraints. The valid action set is then given by

$$\mathcal{A}_{\mathrm{valid}}(s_t) = \{(i,j) : \mathbf{M}(\mathbf{H}_t)_{i,j} = 0\}.$$

Beyond enforcing application-specific structural constraints, action masking also reduces the action space by pruning structurally invalid or undesirable candidates a priori, thereby facilitating more efficient policy learning.

4-cycle masking. It is well known that BP decoding performance is degraded by short cycles in the Tanner graph, particularly 4-cycles. Short cycles undermine the independence of the extrinsic information exchanged during iterative decoding, which in turn degrades decoding performance (Ryan & Lin, 2009; Fan et al., 2008). Accordingly, at each step we identify positions (i,j) that would introduce a new 4-cycle in the Tanner graph of $\mathbf{H}_t$, and mask all such positions from $\mathcal{A}_{\mathrm{valid}}(s_t)$. Specifically, selecting position (i,j) creates a new 4-cycle if there exists a row $i' \neq i$ and a column $j' \neq j$ such that $(\mathbf{H}_t)_{i',j} = (\mathbf{H}_t)_{i,j'} = (\mathbf{H}_t)_{i',j'} = 1$.

For example, consider the following PCM $\mathbf{H}_t$ and its corresponding mask matrix $\mathbf{M}(\mathbf{H}_t)$:

$$\mathbf{H}_t = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{M}(\mathbf{H}_t) = \mathbf{H}_t \vee \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4}$$

where $\vee$ denotes the element-wise OR operation, the first term masks existing edges, and the second term masks positions that would introduce new 4-cycles. This masking prevents exploration of PCMs containing 4-cycles throughout training.

QC masking. To enforce a quasi-cyclic (QC) structure, we partition the PCM $\mathbf{H} \in \{0,1\}^{m\times n}$ into blocks of size $z \times z$, where z denotes the lifting size. When the agent selects an individual edge, all other edges with the same cyclic-shift relation within the corresponding block are automatically added to form a circulant permutation matrix. Edge selections that would violate this block structure are masked out from $\mathcal{A}_{\mathrm{valid}}(s_t)$, thereby preserving the QC structure of the PCM throughout the construction process.

Max-degree masking. The maximum variable-node degree directly determines the hardware complexity of the BP decoder, so applications often require PCMs of the same size under different degree constraints (Richardson et al., 2001). To incorporate this requirement, for a prescribed maximum variable-node degree $d_{\max}$, at each step we identify any variable node v whose current degree has reached $d_{\max}$ and mask the entire corresponding column from $\mathcal{A}_{\mathrm{valid}}(s_t)$. This ensures that every generated PCM satisfies the degree constraint by construction. Moreover, max-degree masking can also be applied at inference time without retraining, as discussed in Section 5.5.

4.3 POLICY OPTIMIZATION IN ZEROCODE

ZeroCode implements the policy network π_θ and value network V_ϕ as two independent MLPs. Both networks take the flattened PCM $\mathrm{vec}(\mathbf{H}_t) \in \{0,1\}^{mn}$ as input. Let f_θ and g_ϕ denote the MLPs that produce the policy logits and the value estimate, respectively. The unmasked policy and the state-value estimate are then given by

$$\pi_\theta(\cdot \mid s_t) = \mathrm{softmax}(f_\theta(\mathrm{vec}(\mathbf{H}_t))), \qquad V_\phi(s_t) = g_\phi(\mathrm{vec}(\mathbf{H}_t)). \tag{5}$$

As shown in the architectural ablation study in Appendix B, the MLP-based actor–critic achieves more stable training and better decoding performance than GNN-based variants. We therefore adopt this MLP architecture in ZeroCode.

The masked policy $\tilde{\pi}_\theta(\cdot \mid s_t)$ is obtained from $\pi_\theta(\cdot \mid s_t)$ by assigning zero probability to all masked actions and renormalizing over $\mathcal{A}_{\mathrm{valid}}(s_t)$, as defined in Eq. (2). Algorithm 1 summarizes the training procedure of ZeroCode. Each episode starts from the all-zero matrix $\mathbf{H}_0 = \mathbf{0}_{m\times n}$, and at every step, ZeroCode samples one valid edge from the masked policy $\tilde{\pi}_\theta$ and adds it to the current PCM. The updated PCM is evaluated by BP decoding, and the resulting transition is stored in the rollout buffer. When an episode terminates, the next episode is initialized from the all-zero matrix. Once the rollout buffer is filled, the policy and value networks are updated by minimizing the PPO loss over the collected transitions. The detailed pseudocode and hyperparameters are provided in Appendix A.

Algorithm 1: Training procedure of ZeroCode

Input: PCM size (m, n), structural constraints
Initialize: π_θ, V_ϕ, and $\mathcal{D} \leftarrow \emptyset$

1: **Start:** Initialize each episode with $\mathbf{H}_0 = \mathbf{0}_{m\times n}$.
2: **Construction:** At step t, construct the action mask $\mathbf{M}(\mathbf{H}_t)$ to reflect the structural constraints. Sample $a_t \sim \tilde{\pi}_\theta(\cdot \mid \mathbf{H}_t)$ from the resulting masked policy, and add the selected edge to obtain $\mathbf{H}_{t+1}$.
3: **Evaluation:** Evaluate $\mathbf{H}_{t+1}$ by BP decoding, retain it in the code library if it is valid, assign R_t according to Eq. (3), and store the transition in $\mathcal{D}$. If the episode terminates, return to the Start step; otherwise, proceed to the next step.
4: **Update:** Once the rollout buffer $\mathcal{D}$ is full, update π_θ and V_ϕ by minimizing L^{PPO} over $\mathcal{D}$, set $\mathcal{D} \leftarrow \emptyset$, and resume the episode from where it was paused.

Output: π_θ

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We consider block lengths $n \in \{32, 64, 128, 256\}$ and a code rate of $r = 0.5$ over the additive white Gaussian noise (AWGN) channel. The maximum number of BP iterations is set to 8 for $n = 32$ and 5 for all other block lengths. During training, the reward is computed based on the BER measured at $\mathrm{E_b/N_0} = 5.0$ dB.

5.2 PERFORMANCE COMPARISON

Figure 3 illustrates the BER performance of ZeroCode across four block lengths. For the $(32, 16)$ code in Figure 3a, ZeroCode is compared against DDPG (Tian et al., 2025), GA (Elkelesh et al., 2019), deep learning (DL) (Choukroun & Wolf, 2024a), classical (Richardson et al., 2001), Random Search, and the CCSDS code (Consultative Committee for Space Data Systems (CCSDS), 2015). The classical baseline is constructed from the degree distribution of Richardson et al. (2001) using the PEG algorithm (Hu et al., 2005). The Random Search baseline starts from the all-zero matrix and adds edges at random positions until a valid PCM is obtained, providing a learning-free reference.

Among these, DDPG (Tian et al., 2025) is the most direct point of comparison, as it shares with ZeroCode the RL-based formulation for PCM construction. In contrast, GA and DL represent independent lines of work based on evolutionary search and differentiable BP, respectively, and serve as reference points for data-driven PCM design. Compared with DDPG, ZeroCode yields a clear improvement of approximately 1 dB at $\mathrm{BER} = 10^{-4}$ for the $(32, 16)$ code, demonstrating the effectiveness of the proposed sequential construction formulation. ZeroCode also outperforms Random Search, suggesting that the learned policy contributes to the observed performance gains.

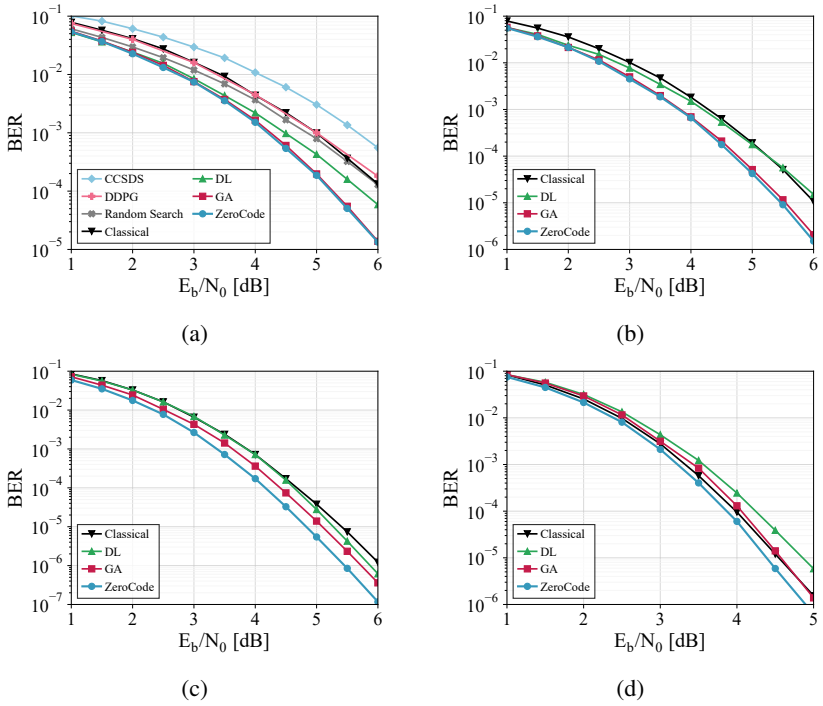


Figure 3: Comparison of the codes constructed by ZeroCode with DDPG (Tian et al., 2025), GA (Elkelesh et al., 2019), DL (Choukroun & Wolf, 2024a), and the classical construction (Richardson et al., 2001) for various code dimensions over the AWGN channel: (a) $(32, 16)$, (b) $(64, 32)$, (c) $(128, 64)$, and (d) $(256, 128)$.

Compared with DL (Choukroun & Wolf, 2024a), ZeroCode achieves substantially better performance across all configurations. Against GA (Elkelesh et al., 2019), ZeroCode achieves comparable performance at $n = 32$ and increasingly outperforms it as the block length grows. Moreover, ZeroCode can directly incorporate diverse structural constraints through masking, as demonstrated in later subsections.

5.3 Component Ablation

Table 1: Ablation study of ZeroCode on the $(32, 16)$ code.

Method	RL Algorithm	Initial PCM	Edge Update	BER ($E_b/N_0 = 5.0$ dB)
(A) (Tian et al., 2025)	DDPG	CCSDS	Multi-Flip	1.0×10^{-3}
(B)	PPO	CCSDS	Multi-Flip	2.1×10^{-3}
(C)	PPO	CCSDS	Single-Flip	1.0×10^{-3}
ZeroCode	**PPO**	**All-zero**	**Single-Add**	$\mathbf{2.2 \times 10^{-4}}$

To examine the contribution of ZeroCode's components, Table 1 compares variants with different RL algorithms, initial PCMs, and edge update schemes. First, comparing (A) and (B) shows that replacing DDPG with PPO alone does not improve performance: with the other components unchanged, the BER increases from 1.0×10^{-3} to 2.1×10^{-3}. Thus, the performance gain of ZeroCode cannot be attributed simply to the choice of PPO.

Next, replacing Multi-Flip in (B) with Single-Flip in (C) reduces the BER from 2.1×10^{-3} to 1.0×10^{-3}. This improvement is consistent with the finer-grained credit assignment enabled by single-edge updates: each reward reflects one edge modification rather than the combined effect of multiple modifications, potentially making individual action effects easier to learn.

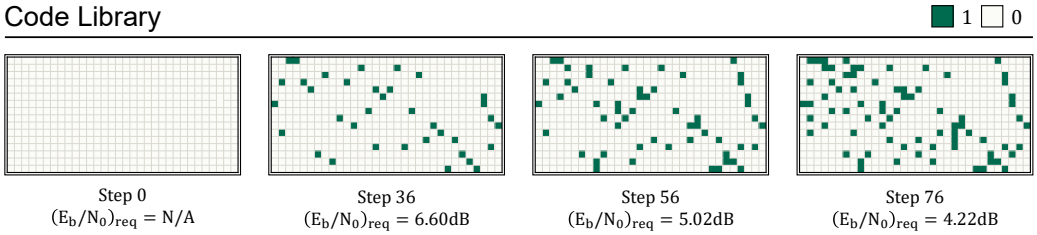


Figure 4: Sequential construction of a code library using a single trained ZeroCode policy for the $(32, 16)$ code. The first matrix shows the all-zero initialization, while the remaining matrices show valid PCMs obtained along the construction trajectory, together with their step indices and the required $(\mathrm{E_b/N_0})_{\mathrm{req}}$ for achieving $\mathrm{BER} = 10^{-3}$.

Finally, comparing (C) with ZeroCode reveals the largest gain among these comparisons. Variant (C) modifies a pre-constructed CCSDS PCM through single-edge flips, whereas ZeroCode constructs a PCM from the all-zero matrix through single-edge additions. This change reduces the BER from 1.0×10^{-3} to 2.2×10^{-4}. These results suggest that ZeroCode's performance improvement over the prior RL-based method primarily stems from reformulating the problem from modifying a pre-constructed PCM to sequentially constructing a PCM from the all-zero matrix.

5.4 A LIBRARY OF CODES FROM A SINGLE ROLLOUT

In this subsection, we show that a single rollout of a trained policy can generate multiple PCMs with different edge counts, forming a code library. Figure 4 visualizes a single rollout of the trained policy for the $(32, 16)$ code. Starting from the all-zero matrix, ZeroCode adds one edge at each step, so the edge count and decoding complexity gradually increase as construction proceeds. At the same time, the required $\mathrm{E_b/N_0}$ to achieve $\mathrm{BER} = 10^{-3}$, denoted by $(\mathrm{E_b/N_0})_{\mathrm{req}}$, generally decreases. The valid PCMs generated along this construction trajectory form a code library with different edge counts.

Figure 5a shows the relationship between the edge count and $(\mathrm{E_b/N_0})_{\mathrm{req}}$ for the resulting code library. As the edge count increases, decoding complexity increases while a lower $(\mathrm{E_b/N_0})_{\mathrm{req}}$ can be achieved, yielding a performance–complexity trade-off curve from a single rollout. Therefore, given an edge-count constraint imposed by an application, an appropriate PCM can be selected from the library without retraining. Moreover, the ZeroCode trade-off curve in Figure 5a lies below the operating points of the existing methods, demonstrating that ZeroCode spans a wide range of edge counts while achieving competitive decoding performance across operating points.

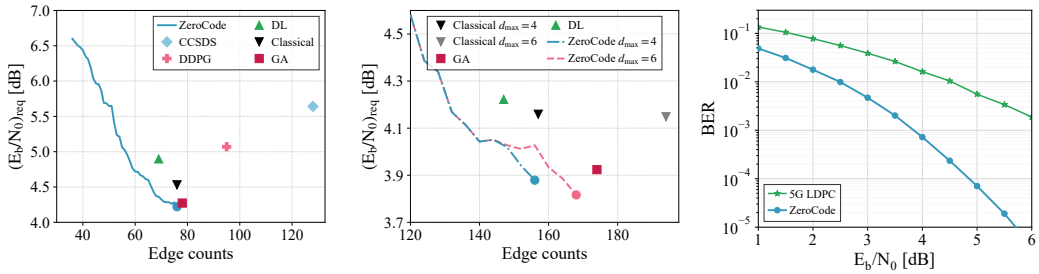
5.5 MAXIMUM DEGREE CONSTRAINT

The max-degree masking introduced in Section 4.2 can also be applied at inference time without retraining. Combined with the code-library construction described in Section 5.4, applying a desired $d_{\max}$ constraint during rollout produces a library of degree-constrained codes spanning a wide range of edge counts.

Figure 5b illustrates this on the $(64, 32)$ code. Starting from a single policy trained with the QC structural mask using lifting factor $z = 4$, but without any maximum-degree constraint, we additionally apply degree masks with $d_{\max} \in \{4, 6\}$ at inference time and plot $(\mathrm{E_b/N_0})_{\mathrm{req}}$ as a function of the edge count. The rollout with $d_{\max} = 6$ extends further than that with the stricter constraint $d_{\max} = 4$, achieving lower $(\mathrm{E_b/N_0})_{\mathrm{req}}$ at the cost of higher edge counts. For $d_{\max} \in \{4, 6\}$, ZeroCode achieves a better performance–complexity trade-off than the conventional approaches—GA (Elkelesh et al., 2019), DL (Choukroun & Wolf, 2024a), and classical (Richardson et al., 2001).

5.6 5G LDPC STRUCTURAL CONSTRAINT

We further demonstrate that ZeroCode can accommodate industry-standard structural constraints through masking. As a representative case, we consider the 5G LDPC code based on BG2 (3GPP,



(a) (b) (c)

Figure 5: (a) Performance–complexity trade-off achieved by a single ZeroCode policy on the $(32, 16)$ code. (b) The same trade-off on the $(64, 32)$ code, obtained by applying degree masks $d_{\max} \in \{4, 6\}$ at inference time to the trained policy. (c) BER comparison between the 5G LDPC code and ZeroCode designed under the raptor-like rate-compatible structural constraint. In (a)–(b), the y-axis represents the required $\mathrm{E_b/N_0}$ to achieve $\mathrm{BER} = 10^{-3}$.

2019), whose base matrix $\mathbf{B}_{5\mathrm{G}}$ follows the raptor-like structure for rate-compatibility

$$\mathbf{B}_{5\mathrm{G}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{0} \\ \mathbf{C} & \mathbf{D} & \mathbf{I} \end{bmatrix}, \tag{6}$$

where $\mathbf{B}$ is a 4×4 dual-diagonal block on the core parity columns, $\mathbf{0}$ is the 4×38 zero block, and $\mathbf{I}$ is the 38×38 identity block on the extension parity columns, all of which are fixed by the standard. The remaining blocks $(\mathbf{A}, \mathbf{C}, \mathbf{D})$ constitute the design region in which the edge pattern can be freely chosen.

ZeroCode incorporates these structural constraints through action masking. The positions in $\mathbf{B}$, $\mathbf{0}$, and $\mathbf{I}$ are permanently masked at every step. Consequently, the valid action set $\mathcal{A}_{\mathrm{valid}}(s_t)$ is restricted to the BG2 design region defined above, and QC masking is applied as well.

Figure 5c compares ZeroCode with the $(100, 20)$ 5G LDPC code (3GPP, 2019) with $z = 2$ on the AWGN channel. While preserving both the raptor-like rate-compatible structure and the QC structure of the 5G LDPC code, ZeroCode achieves a gain of more than 2 dB over the 5G LDPC code at $\mathrm{BER} = 10^{-3}$. This demonstrates that industry-standard structural constraints can be enforced through masking while still allowing high-performing codes to be discovered within the constrained design space.

6 Conclusion

In error-correcting codes, the PCM defines the code, plays a central role in decoding performance, and must often accommodate structural requirements imposed by real-world applications. To address these diverse requirements, we propose ZeroCode, an RL-based framework for flexible code design. ZeroCode formulates PCM construction as a sequential decision-making problem, in which an agent progressively builds a PCM from the all-zero matrix through valid construction actions. Experimental results show that ZeroCode substantially outperforms the prior RL-based approach and achieves superior BER performance over non-RL methods across short-to-medium block lengths.

A key component of ZeroCode is action masking, which enables structural requirements to be enforced directly throughout the construction process. Practically relevant constraints, including 4-cycle-free, quasi-cyclic, maximum-degree, and raptor-like structures, can be incorporated simply by modifying the mask without changing the underlying training procedure. Moreover, a single trained policy naturally generates a library of codes with different edge counts along a single rollout, providing a range of performance–complexity trade-offs without additional training.

In a nutshell, ZeroCode opens a new avenue for on-demand error-correcting code construction by *growing codes from zeros, guiding actions through learning, and enforcing constraints.*

Reproducibility Statement

The training procedure of ZeroCode is summarized in Algorithm 1, and the complete procedure is provided in Algorithm 2 in Appendix A. Appendix A also provides the hyperparameters, reward computation protocol, policy and value network architectures, computational resources, and approximate training cost. Our implementation is available at the anonymous code link provided in the footnote on the first page.

AI Use Statement

Generative AI was used solely as a writing aid during manuscript preparation. Its use was limited to improving wording, readability, and presentation, as well as assisting with LaTeX formatting. All technical content, including the formulation of ZeroCode, experimental design, implementation, and interpretation of the results, was developed independently by the authors. AI-generated suggestions were treated only as editorial input and were checked by the authors before inclusion. The authors remain fully responsible for the accuracy and content of the final manuscript.

References

3GPP. NR; multiplexing and channel coding. Technical Specification TS 38.212, Version 15.6.0, Release 15, 3GPP, July 2019.

H. Chen, X. Dai, H. Cai, W. Zhang, X. Wang, R. Tang, Y. Zhang, and Y. Yu. Large-scale interactive recommendation with tree-structured policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3312–3320, 2019. doi: 10.1609/aaai.v33i01.33013312.

Y. Choukroun and L. Wolf. Error correction code transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 38695–38705, 2022.

Y. Choukroun and L. Wolf. Factor graph optimization of error-correcting codes for belief propagation decoding. *arXiv preprint arXiv:2406.12900*, 2024a.

Y. Choukroun and L. Wolf. A foundation model for error correction codes. In *International Conference on Learning Representations (ICLR)*, 2024b.

Consultative Committee for Space Data Systems (CCSDS). Short block length LDPC codes for TC synchronization and channel coding. Orange Book 231.1-O-1, CCSDS, April 2015.

A. Elkelesh, M. Ebada, S. Cammerer, L. Schmalen, and S. ten Brink. Decoder-in-the-loop: Genetic optimization-based LDPC code design. *IEEE Access*, 7:141161–141170, 2019. doi: 10.1109/ACCESS.2019.2942999.

J. Fan, Y. Xiao, and K. Kim. Design LDPC codes without cycles of length 4 and 6. *Research Letters in Communications*, 2008:354137, 2008. doi: 10.1155/2008/354137.

R. G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1): 21–28, January 1962. doi: 10.1109/TIT.1962.1057683.

S. Habib, A. Beemer, and J. Kliewer. Learning to decode: Reinforcement learning for decoding of sparse graph-based channel codes. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 22396–22406, 2020.

S. Habib, A. Beemer, and J. Kliewer. RELDEC: Reinforcement learning-based decoding of moderate length LDPC codes. *IEEE Transactions on Communications*, 71(10):5661–5674, October 2023. doi: 10.1109/TCOMM.2023.3296621.

G. He, J.-C. Belfiore, I. Land, G. Yang, X. Liu, Y. Chen, R. Li, J. Wang, Y. Ge, R. Zhang, and W. Tong. β-Expansion: A theoretical framework for fast and recursive construction of polar codes. In *IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, December 2017. doi: 10.1109/GLOCOM.2017.8254146.

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

L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang. AI coding: Learning to construct error correction codes. *IEEE Transactions on Communications*, 68(1):26–39, January 2020. doi: 10.1109/TCOMM.2019.2951403.

S. Huang and S. Ontañón. A closer look at invalid action masking in policy gradient algorithms. In *Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, volume 35, 2022. doi: 10.32473/flairs.v35i.130584.

Huawei Technologies Co., Ltd. and HiSilicon Technologies Co., Ltd. Polar code design and rate matching. Technical Report R1-167209, 3rd Generation Partnership Project (3GPP), Gothenburg, Sweden, August 2016. Meeting #86.

H.-Y. Kwak, D.-Y. Yun, Y. Kim, S.-H. Kim, and J.-S. No. Boosting learning for LDPC codes to improve the error-floor performance. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 22115–22131, 2023.

Hee-Youl Kwak, Dae-Young Yun, Yongjune Kim, Sang-Hyo Kim, and Jong-Seon No. Boosted neural decoders: Achieving extreme reliability of LDPC codes for 6G networks. *IEEE Journal on Selected Areas in Communications*, 43(4):1089–1102, April 2025. doi: 10.1109/JSAC.2025.3531553.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, March 1999. doi: 10.1109/18.748992.

A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nova, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean. A graph placement methodology for fast chip design. *Nature*, 594 (7862):207–212, June 2021. doi: 10.1038/s41586-021-03544-w.

E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, February 2018. doi: 10.1109/JSTSP.2017.2788405.

OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

S.-J. Park, H.-Y. Kwak, S.-H. Kim, S. Kim, Y. Kim, and J.-S. No. Multiple-masks error correction code transformer for short block codes. *IEEE Journal on Selected Areas in Communications*, 43 (7):2518–2529, 2025a. doi: 10.1109/JSAC.2025.3559154.

S.-J. Park, H.-Y. Kwak, S.-H. Kim, Y. Kim, and J.-S. No. CrossMPT: Cross-attention message-passing transformer for error correcting codes. In *International Conference on Learning Representations (ICLR)*, 2025b.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

V. Ramkumar, S. B. Balaji, B. Sasidharan, M. Vajha, M. N. Krishnan, and P. V. Kumar. Codes for distributed storage. *Foundations and Trends in Communications and Information Theory*, 19(4): 547–813, 2022. doi: 10.1561/0100000115.

T. Richardson and S. Kudekar. Design of low-density parity-check codes for 5G New Radio. *IEEE Communications Magazine*, 56(3):28–34, March 2018. doi: 10.1109/MCOM.2018.1700839.

T. Richardson, M. A. Shokrollahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, February 2001. doi: 10.1109/18.910578.

W. E. Ryan and S. Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, 2009. ISBN 978-0-521-84868-8. doi: 10.1017/CBO9780511803253.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

M. Shirvanimoghaddam, M. S. Mohammadi, R. Abbas, A. Minja, C. Yue, B. Matuz, G. Han, Z. Lin, W. Liu, Y. Li, S. J. Johnson, and B. Vucetic. Short block-length codes for ultra-reliable low latency communications. *IEEE Communications Magazine*, 57(2):130–137, February 2019. doi: 10.1109/MCOM.2018.1800181.

P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52 (4):R2493–R2496, October 1995. doi: 10.1103/PhysRevA.52.R2493.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

K. Tian, C. Yue, C. She, B. Vucetic, and Y. Li. GNN-based auto-encoder for short linear block codes: A DRL approach. *IEEE Transactions on Communications*, 73(10):8558–8573, October 2025. doi: 10.1109/TCOMM.2025.3565602.

P. Trifonov. Efficient design and decoding of polar codes. *IEEE Transactions on Communications*, 60(11):3221–3227, November 2012. doi: 10.1109/TCOMM.2012.081512.110872.

C. Yue, V. Miloslavskaya, M. Shirvanimoghaddam, B. Vucetic, and Y. Li. Efficient decoders for short block length codes in 6G URLLC. *IEEE Communications Magazine*, 61(4):84–90, April 2023. doi: 10.1109/MCOM.001.2200275.

K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang. LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*, pp. 243–256, 2013.

A TRAINING OF ZEROCODE

A.1 TRAINING PROCEDURE

Algorithm 2: Training procedure of ZeroCode

Input: PCM dimensions (m, n), structural constraints, rollout size N, mini-batch size b, epochs K, total timesteps T
Output: Trained policy π_θ
Initialize θ and ϕ; set $\mathcal{D} \leftarrow \emptyset$, $\mathbf{H}_t \leftarrow \mathbf{0}_{m\times n}$, and global step $\leftarrow 0$
while *global step* $< T$ **do**
  /* Rollout collection */
  while $|\mathcal{D}| < N$ **do**
    Construct the action mask $\mathbf{M}(\mathbf{H}_t)$ to enforce the given structural constraints, as described in Section 4.2
    Sample $a_t \sim \tilde{\pi}_\theta(\cdot \mid \mathbf{H}_t)$ from the masked policy in Eq. (2)
    Apply the sampled action to $\mathbf{H}_t$ to obtain $\mathbf{H}_{t+1}$
    compute $R_t = \Lambda(\mathbf{H}_{t+1})$ as in Eq. (3)
    Set done_t to true when s_{t+1} has no valid actions, and false otherwise
    Store $(\mathbf{H}_t, a_t, R_t, \mathbf{H}_{t+1}, \mathrm{done}_t)$ in $\mathcal{D}$
    global step $+= 1$
    if done_t **then**
      $\mathbf{H}_t \leftarrow \mathbf{0}_{m\times n}$
    else
      $\mathbf{H}_t \leftarrow \mathbf{H}_{t+1}$
    end
  end
  /* Policy and value update */
  Compute GAE advantages $\{\hat{A}_t\}$ and target returns from $\mathcal{D}$
  Update (θ, ϕ) for K epochs using mini-batches of size b according to Eq. (7)
  $\mathcal{D} \leftarrow \emptyset$
end
return π_θ

Algorithm 2 summarizes the complete training procedure of ZeroCode. Each episode starts from the all-zero PCM $\mathbf{H}_0 = \mathbf{0}_{m\times n}$. At step t, the action mask $\mathbf{M}(\mathbf{H}_t)$ is constructed, and one valid edge is sampled from $\tilde{\pi}_\theta(\cdot \mid \mathbf{H}_t)$ and added to the current PCM. The resulting PCM $\mathbf{H}_{t+1}$ is then evaluated by the BP decoder. The resulting reward and transition are appended to the rollout buffer. The episode terminates when no valid edge remains in the masked action set. The environment is then reset.

Rollout collection continues across episodes until the buffer contains N transitions. We then compute target returns and generalized-advantage estimates and update the policy and value networks for K optimization epochs using mini-batches under the PPO objective. The buffer is cleared after each update. This process is repeated until the total number of training timesteps reaches T, yielding the trained policy π_θ.

A.2 OPTIMIZATION DETAILS

Table 2 reports the optimization settings used in all experiments. For each PPO update, we collect $2{,}048$ transitions and perform 10 optimization epochs with mini-batches of 64. We use a learning rate of 10^{-5} for both the policy and value networks. Advantages are estimated with GAE (Schulman et al., 2016), using discount factor $\gamma = 0.99$ and $\lambda = 0.95$.

The training objective combines the clipped policy loss, the value-function loss, and an entropy bonus:

$$L_{\mathrm{PPO}}(\theta, \phi) = \hat{\mathbb{E}}_t \left[-L_t^{\mathrm{CLIP}}(\theta) + c_1 L_t^{\mathrm{VF}}(\phi) - c_2 S[\tilde{\pi}_\theta](s_t) \right]. \tag{7}$$

Table 2: Hyperparameters used to train ZeroCode.

Parameter	**Value**
Learning rate	10^{-5}
Total timesteps T	3×10^6
Rollout size N	2,048
Number of epochs K	10
Mini-batch size b	64
Discount factor γ	0.99
GAE parameter λ	0.95
Clip range ϵ	0.2
Value loss coefficient c_1	0.5
Entropy coefficient c_2	0.01
Max gradient norm	0.5
Target KL divergence	0.02
Policy hidden layers π_θ	$[4096,\ 2048]$
Value hidden layers V_ϕ	$[2048,\ 1024]$

We set the clipping parameter to $\epsilon = 0.2$, the value-loss coefficient to $c_1 = 0.5$, and the entropy coefficient to $c_2 = 0.01$. For optimization stability, the gradient norm is clipped at 0.5. We also use a target KL divergence of 0.02 and terminate the remaining epochs of an update when this threshold is exceeded.

Networks and compute. The policy π_θ and value function V_ϕ are represented by separate two-layer MLPs. Their hidden-layer sizes are $[4096, 2048]$ and $[2048, 1024]$, respectively. We use the MaskablePPO implementation in Stable-Baselines3 Contrib v2.7.1 (Raffin et al., 2021), which supports invalid-action masking. All experiments run on one NVIDIA RTX A5000 GPU with 24 GB of memory; depending on the code configuration, training one policy takes approximately 4–18 hours.

A.3 Training cost

Table 3: Approximate wall-clock training cost for the $(32, 16)$ code. The code library contains the $L = 41$ valid PCMs from the first valid design at 36 edges to the best design at 76 edges. For GA and DL, library costs are estimated as L times the single-PCM cost, assuming independent optimization for each target edge count.

Method	Cost for a single PCM	Cost for a library ($L = 41$)
ZeroCode	4 hours (t_a)	4 hours (t_a)
GA (Elkelesh et al., 2019)	6 hours (t_b)	246 hours (Lt_b)
DL (Choukroun & Wolf, 2024a)	2 hours (t_c)	82 hours (Lt_c)

For the $(32, 16)$ code, the GA runtime was measured using its public implementation and is comparable to that of ZeroCode because both methods repeatedly evaluate candidate PCMs with a BP decoder. The DL method has a shorter optimization time, but yielded lower BER performance in our experiments.

For a single target PCM, ZeroCode has a training cost comparable to GA. Under an independent optimization protocol for each target edge count, the estimated costs of constructing an $L = 41$ code library are Lt_b for GA and Lt_c for DL. ZeroCode instead obtains all $L = 41$ valid intermediate PCMs along a rollout of a single trained policy, providing multiple performance–complexity operating points without retraining.

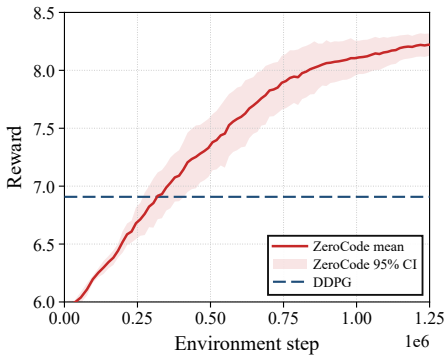


Figure 6: Training robustness of ZeroCode across eight random seeds for the $(32, 16)$ code. The solid red curve shows the mean best-PCM reward $R_{\text{best}} = -\ln(\text{BER}_{\text{best}})$ across seeds. The shaded region denotes the 95% confidence interval across eight random seeds. Rewards are evaluated at $\text{E}_\text{b}/\text{N}_0 = 5.0$ dB and averaged over non-overlapping bins of 200 episodes. The horizontal dashed line denotes the reward of the final PCM produced by DDPG (Tian et al., 2025) under the same evaluation setting.

A.4 Detailed Experimental Settings and Statistical Stability

MLP architecture. The MLP implementation follows the default implementation choices of Stable-Baselines3 Contrib. Specifically, Tanh is used as the activation function for the hidden layers, Adam is used as the optimizer, and the network weights are initialized using orthogonal initialization.

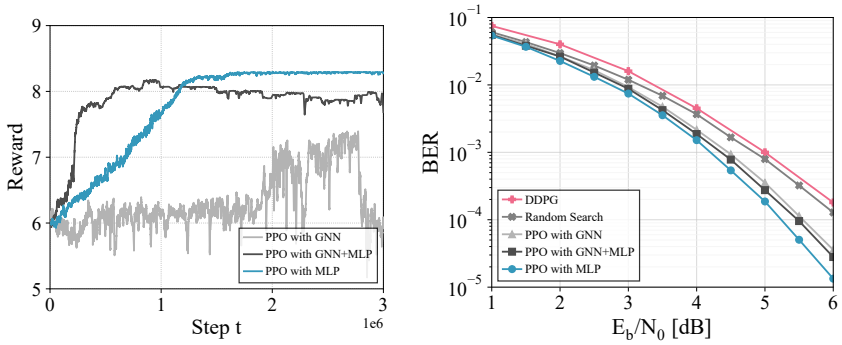
GNN architecture. For the GNN-based architectures evaluated in Appendix B, we follow the GNN configuration of Tian et al. (2025). Each PCM element is treated as a node in a lattice graph and exchanges messages with its four neighboring elements. The embedding and message dimensions are both set to 10. The GNN consists of 3 message-passing layers, and the MLPs used for message computation and embedding updates consist of 3 layers with 40 hidden units.

Channel model. Each codeword $\mathbf{c} \in \{0, 1\}^n$ is BPSK-modulated as $\mathbf{x} = \mathbf{1} - 2\mathbf{c}$ and transmitted over an AWGN channel, yielding $\mathbf{y} = \mathbf{x} + \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ and $\sigma^2 = 1/\left(2r\,\text{E}_\text{b}/\text{N}_0\right)$. The resulting channel LLRs, $\mathbf{m}^{\text{ch}} = 2\mathbf{y}/\sigma^2$, are provided as input to the BP decoder. The polar-code experiment in Figure 8(b) instead uses QPSK modulation, with channel conditions specified in terms of $\text{E}_\text{s}/\text{N}_0$.

Reward and evaluation. Since ZeroCode sequentially constructs the PCM from the all-zero matrix, the decoding performance of intermediate PCMs is also used as the reward signal during training. For all PCMs encountered along the construction trajectory, we fix the code rate to the target rate $r = k/n$, regardless of the current rank of $\mathbf{H}_t$. During training, the error rate used to compute the reward is estimated by continuing the simulation until 100 frame errors are observed. For the final performance curves, the simulation at each evaluated SNR point is continued until 1,000 frame errors are observed.

Baseline implementation. For DDPG (Tian et al., 2025), we directly use the PCMs provided by the publicly available implementation. For GA (Elkelesh et al., 2019), we use the publicly available implementation and follow its original optimization procedure. Specifically, the 20 best-performing PCMs in each population are retained as parents for mutation and crossover, and the optimization is performed for 500 generations. For DL (Choukroun & Wolf, 2024a), we perform 20 optimization steps, with the line search at each step restricted to the 110 smallest candidate step sizes.

Statistical stability. To assess sensitivity to random initialization, we independently train ZeroCode with eight random seeds for the $(32, 16)$ code. As shown in Figure 6, the ZeroCode confidence interval rises above the DDPG reference after the initial exploration phase, and all eight runs end above this reference. These results indicate stable training behavior across random seeds.



(a) (b)

Figure 7: Comparison of three actor–critic architectures (GNN, GNN+MLP, and MLP) on the $(32, 16)$ code. (a) Training curves show the 100-episode moving average of the reward $R_{\text{best}} = -\ln(\text{BER}_{\text{best}})$ of the best PCM encountered in each episode. (b) Performance of the resulting codes constructed by each architecture.

B Actor–critic architecture

The most closely related RL-based method, Tian et al. (2025), employs a GNN-based actor–critic architecture. Since PCM construction is naturally associated with graph structures, incorporating a graph-based representation may appear well motivated. However, it remains unclear whether an explicit graph embedding is necessary for the sequential PCM construction problem considered in ZeroCode. To investigate this, we compare three actor–critic architectures with different state representations: (i) **GNN**, in which a GNN directly performs both state embedding and policy/value estimation; (ii) **GNN+MLP**, in which a GNN first embeds the current state and the resulting embedding is processed by an MLP-based actor–critic; and (iii) **MLP**, in which the current PCM is flattened and fed directly into an MLP without any learnable graph embedding as shown in Section 4.3.

Figure 7a compares the training curves of the three architectures on the $(32, 16)$ code using the best-PCM reward R_{best}. The GNN model remains at a low reward throughout training, suggesting that the GNN representation may be less effective for the fine-grained action selection required in this setting. The GNN+MLP model achieves faster reward growth during the early stage of training, indicating that graph-based embeddings help stabilize early exploration; however, its performance saturates relatively early. In contrast, the MLP model improves more steadily and ultimately achieves the highest reward. The same trend is observed in Figure 7b, where the MLP-based architecture achieves the best BER performance among the three.

These results suggest that an explicit graph embedding is not necessary for ZeroCode. Since the current PCM $\mathbf{H}_t$ already preserves the positional information of candidate edges, the MLP-based actor–critic can learn the relevant structural patterns directly from the flattened input. Accordingly, ZeroCode adopts the MLP-based architecture.

C Action masking ablation

To compare different action masking strategies, we independently train policies with 4-cycle, maximum-degree, and QC masking for rate-$1/2$ codes with block lengths $n \in \{32, 64, 128, 256\}$. QC masking is considered only for $n \geq 64$, given the small matrix size at $n = 32$. For each policy, the best-performing PCM is evaluated at $\text{E}_\text{b}/\text{N}_0 = 5.0$ dB using BP decoding with eight iterations for $n = 32$ and five iterations for the other block lengths. Table 4 summarizes the resulting BER, edge count, maximum variable-node degree, and numbers of 4- and 6-cycles.

The results confirm that the constraints imposed by each action mask are satisfied as intended: 4-cycle masking eliminates all 4-cycles, maximum-degree masking produces PCMs with $d_{\max} \leq 8$, and QC masking preserves the prescribed quasi-cyclic structure. QC masking achieves the lowest BER for

Table 4: Comparison of action masking strategies. BER is evaluated at $\mathrm{E_b/N_0} = 5.0$ dB using a BP decoder; the lowest BER for each block length is shown in bold.

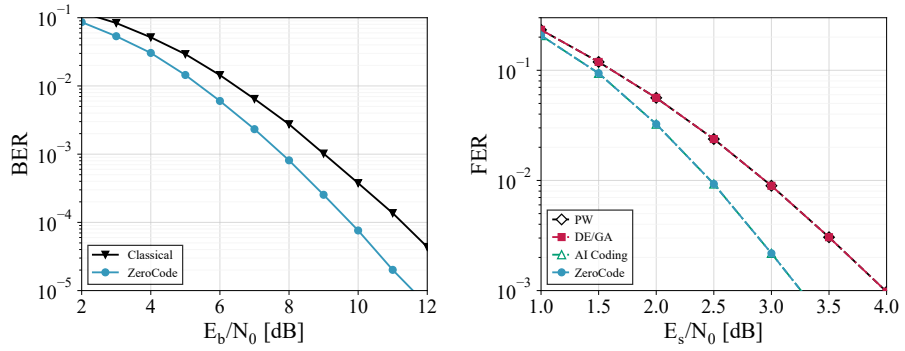
(n, k)	Action mask	Edges	$d_{\max}$	4-cycle	6-cycle	BER
$(32, 16)$	4-cycle	76	5	0	88	$\mathbf{1.81 \times 10^{-4}}$
	Max-degree ($d_{\max} = 8$)	76	5	3	99	2.24×10^{-4}
$(64, 32)$	4-cycle	178	6	0	311	6.76×10^{-5}
	Max-degree ($d_{\max} = 8$)	176	7	5	296	7.25×10^{-5}
	QC ($Z = 8$)	168	4	0	144	$\mathbf{4.23 \times 10^{-5}}$
$(128, 64)$	4-cycle	400	8	0	818	1.15×10^{-5}
	Max-degree ($d_{\max} = 8$)	394	6	14	572	1.81×10^{-5}
	QC ($Z = 16$)	368	4	0	192	$\mathbf{5.44 \times 10^{-6}}$
$(256, 128)$	4-cycle	864	7	0	870	2.29×10^{-6}
	Max-degree ($d_{\max} = 8$)	833	7	21	705	3.14×10^{-6}
	QC ($Z = 16$)	848	5	0	720	$\mathbf{6.49 \times 10^{-7}}$

$n \in \{64, 128, 256\}$, while 4-cycle masking performs best among the evaluated strategies for $n = 32$. Accordingly, the ZeroCode performance curves in Figure 3 use the best-performing masking strategy for each block length: 4-cycle masking for $n = 32$ and QC masking for the remaining block lengths.

D GENERALITY

The main experiments focus on PCM construction with BP decoding over the AWGN channel. To examine whether ZeroCode extends beyond this channel and code–decoder combination, we conduct two auxiliary experiments involving a different channel model and a different code family.

First, we evaluate the $(32, 16)$ LDPC code constructed by ZeroCode over a Rayleigh fading channel and compare it with the classical $(32, 16)$ LDPC code used in the AWGN experiment. As shown in Figure 8(a), ZeroCode achieves a lower BER than the classical construction throughout the evaluated $\mathrm{E_b/N_0}$ range, indicating that its performance gain extends beyond the AWGN channel.



(a) (b)

Figure 8: Generality of ZeroCode: (a) BER comparison between ZeroCode and the classical LDPC construction over a Rayleigh fading channel; (b) FER comparison of the ZeroCode polar-code construction with AI Coding, PW, and DE/GA over the AWGN channel.

Second, we apply ZeroCode's sequential decision-making principle to polar-code construction by treating the selection of information-bit positions as a sequence of discrete actions. ZeroCode uses PPO to learn a policy for constructing the polar information set. The state is a length-n binary vector whose ones denote the selected information-bit positions. At each step, an action selects one of the remaining frozen positions and sets the corresponding state entry to one. For a target (n, k) code,

the episode terminates after k positions have been selected. The policy is trained using the negative logarithm of the frame error rate (FER) measured at $\mathrm{E_s}/\mathrm{N_0} = 3.0$ dB as the reward.

We compare ZeroCode with AI Coding (Huang et al., 2020), polarization weight (PW) (Huawei Technologies Co., Ltd. & HiSilicon Technologies Co., Ltd., 2016; He et al., 2017), and density evolution/Gaussian approximation (DE/GA) (Trifonov, 2012). All methods are evaluated on a $(128, 64)$ polar code with QPSK modulation over the AWGN channel using the same path-metric-based successive-cancellation list (SCL-PM) decoder with list size 8. For this configuration, PW and DE/GA select the same information-bit set and therefore yield identical FER performance. As shown in Figure 8(b), ZeroCode matches the reported FER performance of AI Coding, and both outperform PW and DE/GA over the evaluated $\mathrm{E_s}/\mathrm{N_0}$ range. Together, these auxiliary results demonstrate the applicability of ZeroCode's sequential construction approach beyond a single channel model or code–decoder combination.

E LIMITATIONS

This work primarily focuses on PCM construction with BP decoding. While Appendix D demonstrates preliminary generalization to a different channel model and to polar-code construction, the broader applicability of ZeroCode has not yet been systematically evaluated. Extending and validating the framework across diverse decoders, channels, and code representations is an important direction for future work.