Policy-Guided MCTS for near Maximum-Likelihood Decoding of Short Codes

Yuan Tian[1], Chentao Yue[1], Peng Cheng[2], Gaoyang Pang[1], Branka Vucetic[1], and Yonghui Li[1]
[1]School of Electrical and Computer Engineering, The University of Sydney, Australia
[2]Department of Computer Science and Information Technology, La Trobe University, Australia
Email: {yuan.tian1, chentao.yue, gaoyang.pang, branka.vucetic, yonghui.li}@sydney.edu.au, p.cheng@latrobe.edu.au

***Abstract*—In this paper, we propose a policy-guided Monte Carlo Tree Search (MCTS) decoder that achieves near maximum-likelihood decoding (MLD) performance for short block codes. The MCTS decoder searches for test error patterns (TEPs) in the received information bits and obtains codeword candidates through re-encoding. The TEP search is executed on a tree structure, guided by a neural network policy trained via MCTS-based learning. The trained policy guides the decoder to find the correct TEPs with minimal steps from the root node (all-zero TEP). The decoder outputs the codeword with maximum likelihood when the early stopping criterion is satisfied. The proposed method requires no Gaussian elimination (GE) compared to ordered statistics decoding (OSD) and can reduce search complexity by 95% compared to non-GE OSD. It achieves lower decoding latency than both OSD and non-GE OSD at high SNRs.**

I. Introduction

Ultra-reliable low-latency communications (URLLC) has been defined by the third generation partnership project (3GPP) as a key service categories in beyond-5G and 6G networks [1]. URLLC requires end-to-end latency below 1 ms and block error rate (BLER) of 10^{-5}–10^{-7} [2]. To satisfy these requirements, URLLC will demand short channel codes with low decoding complexity and superior BLER performance [2], [3]. Among existing solutions, low-density parity-check (LDPC) codes with belief propagation (BP) decoding offer low complexity, but suffer significant BLER performance degradation at short block length [2]. High-density codes such as Bose-Chaudhuri-Hocquenghem (BCH) codes can approach the finite-length bound under near maximum likelihood decoding (MLD) [3]; however, the exponential decoding complexity hinders their application. Recently, deep learning (DL) has been applied to channel coding and decoding, with extensive work on enhancing BP decoding via deep neural networks (DNNs), including neural belief propagation (NBP) [4], [5], as well as joint optimization of codes and BP decoders [6], [7]. These methods show the potential of DL to enhance existing decoders. Nevertheless, BP and its DL-based variants rely on the Tanner graph of the code. They remain fundamentally limited and exhibit a persistent performance gap from MLD at short block lengths. Short codes typically have dense parity-check matrices that induce short cycles in code Tanner graphs, causing correlation among messages and degraded BP decoding performance.

List-based decoding offers an alternative to achieve near-MLD. These decoders generate a candidate list of codewords and identify the one with the minimum Euclidean distance to the received signal. Representatives include Chase decoding [8] and ordered statistics decoding (OSD) [9]. OSD re-orders the columns of the code generator matrix based on the reliability of the received bits, then transforms it into systematic form via Gaussian elimination (GE), which places the most reliable bits in the information set. Then it applies test error patterns (TEPs) to the information set to obtain codeword candidates. An order-m OSD searches $O(k^m)$ TEPs with message length k [10]. Recent work has also proposed non-GE OSD variants [11], which omit the GE overhead but require examining more TEPs, particularly at low SNRs.

Existing efforts to reduce OSD complexity mainly focus on early stopping criterion [11]–[13] and discarding unnecessary TEPs [11], [14], [15]. Nevertheless, decoding efficiency also heavily depends on the TEP search strategy [16]. Existing decoders typically search TEPs in ascending order of Hamming weight, as fewer errors are more probable. However, this strategy often requires examining many TEPs before finding the correct TEP with large Hamming weight. Monte Carlo Tree Search (MCTS) offers a promising approach to address this TEP search problem. MCTS has achieved remarkable success in complex domains such as Go, where AlphaGo [17] and AlphaZero [18] reached superhuman performance by combining MCTS with DNNs. In these systems, a learned policy guides the search toward promising regions, while MCTS balances exploration and exploitation through statistical simulations during training and inference.

In this paper, we develop a policy-guided MCTS decoder that achieves near-MLD performance for short codes. Instead of enumerating TEPs by ascending Hamming weight, our proposed method organizes the TEP space as a deterministic binary tree and traverse it using depth-first search. A neural network policy, trained via MCTS, guides branch selection during the decoding. For training, MCTS explores the tree guided by the current policy network, and the statistics from successful trajectories are used to refine the policy network iteratively. Once trained, the policy directs the search toward the correct TEP along the shortest path. Experiments show that our approach requires significantly fewer TEP searches compared to non-GE OSD. For the $(48, 24)$ code at 1 dB, our decoder needs fewer than 200 TEPs versus over 5000 for non-GE OSD to find MLD results. It also achieves lower decoding time than OSD at high SNRs, as it avoids GE. The proposed search framework can be generalized to any OSD variants and

The work of Chentao Yue was supported by ARC DECRA under Grant DE250101332. Code available: https://github.com/1014Tea/MCTS-Decoding.

work with any existing early stopping or TEP pruning method.

The rest of this paper is organized as follows. Section II introduces the preliminaries. Section III presents the proposed MCTS-guided decoding algorithm. The simulation results are discussed in Section IV, and Section V concludes the paper.

II. Preliminaries

A. Linear Block Codes and Channel Model

A binary linear block code $\mathcal{C}(n,k)$ is defined by a generator matrix $\mathbf{G} \in \{0,1\}^{k\times n}$, where n and k denote the length of the codeword and the message, respectively. The codeword $\mathbf{c}$ is obtained as $\mathbf{c} = \mathbf{bG}$, where $\mathbf{b} \in \{0,1\}^k$ is the binary message. The codeword $\mathbf{c} = [c_1, \ldots, c_n]$ is transmitted over the additive white Gaussian noise (AWGN) channel with binary phase shift keying (BPSK) modulation. Let $\mathbf{x} = 1 - 2\mathbf{c} \in \{-1,1\}^n$ be the modulated symbol of $\mathbf{c}$ and $\mathbf{z}$ be the AWGN vector with noise power σ^2, i.e., $z_i \sim \mathcal{N}(0,\sigma^2)$. The received signal $\mathbf{r} = [r_1, \ldots, r_n]$ is obtained as $\mathbf{r} = \mathbf{x} + \mathbf{z}$. We define the SNR in decibels as $\gamma = 10\log_{10}\big(\frac{1}{\sigma^2}\big)$.

B. Ordered Statistics Decoding and its non-GE variant

OSD provides a practical approach to near-MLD performance. Let log-likelihood ratios (LLRs) for each received bit be $\ell_i = \ln\frac{\Pr(c_i=0|r_i)}{\Pr(c_i=1|r_i)} = \frac{2r_i}{\sigma^2}$, for $i = \{1,\ldots,n\}$. OSD performs a column permutation π_{OSD} on the generator matrix $\mathbf{G}$ according to the descending absolute LLR values $|\ell_i|$, such that columns corresponding to more reliable bits are placed in the first k positions. GE is then applied to the permuted generator matrix to obtain a systematic form $\mathbf{G}' = [\mathbf{I}_k \mid \mathbf{P}]$, where $\mathbf{I}_k$ is a $k \times k$ identity matrix that corresponds to the k most reliable bits. Let $\mathbf{b'_0} \in \{0,1\}^k$ denote the hard-decision vector of the k most reliable bits, referred to as the most reliable basis (MRB). A TEP $\mathbf{e} \in \{0,1\}^k$ with Hamming weight $w(\mathbf{e})$ modifies the MRB via $\mathbf{b'_e} = \mathbf{b'_0} \oplus \mathbf{e}$. Then, a candidate codeword $\mathbf{c_e}$ is generated by re-encoding $\mathbf{b'_e}$ according to $\mathbf{c'_e} = \mathbf{b'_e}\,\mathbf{G}'$, and applying the inverse permutation $\mathbf{c_e} = \pi_{\mathrm{OSD}^{-1}}(\mathbf{c'_e})$. An order-$m$ OSD searches only TEPs with Hamming weight $w(\mathbf{e}) \le m$, which are typically enumerated in ascending weight order, as fewer bit errors are more likely [9]. Therefore, the total number of TEPs is $\sum_{i=0}^{m}\binom{k}{i} = O(k^m)$. Finally, the decoder outputs the candidate codeword with the minimum Euclidean distance $d(\mathbf{c_e}, \mathbf{r})$ to the received signal, where $d(\mathbf{c_e}, \mathbf{r}) = \|\mathbf{r} - (1 - 2\mathbf{c_e})\|_2$.

The non-GE OSD variant [11] avoids both the column permutation and Gaussian elimination steps. Instead, it applies TEPs directly to the hard decisions of the first k received bits and encodes all candidates via the original generator $\mathbf{G}$. Specifically, let $\mathbf{b}_0$ denote the hard-decision vector of the first k position of $\mathbf{r}$. Non-GE OSD directly obtain a codeword candidates by $\mathbf{c_e} = (\mathbf{b_0} \oplus \mathbf{e})\mathbf{G}$. At high SNRs, OSD often finds the correct TEP within a few search steps and can terminate early using stopping rules [11] or cyclic redundancy check (CRC). In this regime, the GE overhead dominates the total cost, making non-GE OSD more efficient [11]. At low SNRs, however, non-GE OSD requires a much higher decoding order m to achieve the same BLER as the original OSD, since the fixed information set is less reliable than MRB. Prior work has addressed the low-SNR inefficiency of non-GE OSD through adaptive GE reduction [11], affine-invariant permutations [19], and staircase generator matrices [20].

In this paper, we employ the policy-guided MCTS to reduce the TEP search complexity in non-GE OSD. Notably, the proposed framework can be readily extended to GE-based OSD and other decoder variants involving TEP search.

C. Monte Carlo Tree Search

Following AlphaZero, MCTS can be viewed as a self-play reinforcement learning procedure [18]. MCTS iteratively balances exploration and exploitation during tree search through four steps [17] (See Fig. 2).

1) Selection: Starting from the root, repeatedly choose a child according to a selection policy (e.g., PUCT with DNN [18], see Section III-B), descending the current tree until reaching a node that can be further expanded to children.

2) Expansion: Upon reaching a node that can be further expanded, add one or a set of new child nodes to the tree, thereby growing the search frontier.

3) Simulation (Evaluation): From the newly added node, perform a simulation to evaluate the *node value*, by using a neural network or random selection to quickly expand the tree into deeper levels and estimate future potential outcomes.

4) Backpropagation: Propagate the estimated value back along the path to the root, updating the statistics (visit counts and accumulated values) at each visited node. These updated statistics guide future selection steps and policy training.

These four steps are executed iteratively for multiple iterations. Unlike AlphaZero, in this paper, we can leverage the ground truth (e.g., MLD codeword) for training. MCTS explores the TEP tree to identify successful search paths, generating high-quality supervisory trajectories to train the policy network. The improved policy subsequently biases MCTS toward correct decoding paths with higher probability in the subsequent training iterations. This iterative process, where the algorithm self-generates training supervision, represents a form of self-supervised learning [21]. Unlike conventional supervised learning, MCTS training requires no optimal-path labels, yielding stronger generalization across different received noisy signals (see Section III).

III. Policy-Guided MCTS Decoding

A. Tree Structure of TEPs and MCTS Formulation

Inspired by [22], we build a binary tree for TEPs as follows.

Definition 1 (TEP Tree)**.** A TEP tree is a deterministic binary tree that enumerates TEPs $\mathbf{e} \in \{0,1\}^k$ with Hamming weight $w(\mathbf{e}) \le m$, where $1 \le m \le k$. Each node corresponds to a TEP $\mathbf{e}$, represented by the strictly increasing index set of its non-zero positions

$$\mathcal{Z}_{\mathbf{e}} = \{z_1, z_2, \ldots, z_\ell\},$$

where $z_1 < z_2 < \cdots < z_\ell$ and $\ell = w(\mathbf{e})$. The root of the tree is defined as the all-zero TEP with $\mathcal{Z}_{\mathbf{0}} = \varnothing$. From any node $\mathcal{Z}_{\mathbf{e}}$, at most two children are generated:

- Extended child $\mathcal{Z}^{\mathrm{ext}}_{\mathbf{e}} = \{z_1, \ldots, z_\ell,\, k\}$.
- Adjacent child $\mathcal{Z}^{\mathrm{adj}}_{\mathbf{e}} = \{z_1, \ldots, z_{\ell-1},\, z_\ell - 1\}$.

Children are created only when feasible. The extended child requires $z_\ell \neq k$, and the adjacent child requires $z_\ell - 1 \neq z_{\ell-1}$.

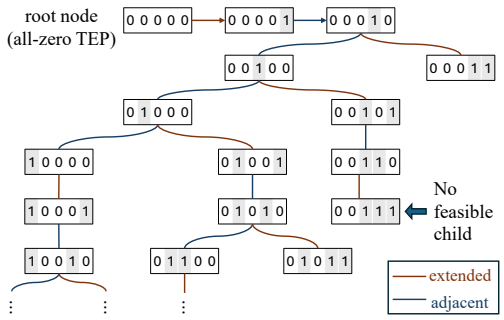


Fig. 1: The TEP tree for $k = 5$ and $m = 3$.

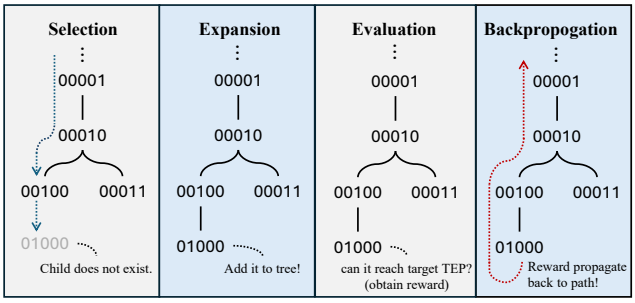


Fig. 2: An MCTS iteration on a TEP tree. MCTS reaches TEP 00100, selects an unexpanded child 01000, then expands, evaluates, and backpropagates.

According to Definition 1, a TEP tree is fully determined by message length k and decoding order m, and it enumerates all TEPs with $w(\mathbf{e}) \leq m$ exactly once. Figure 1 shows the TEP tree for $k = 5$ and $m = 3$. For the root node $\mathcal{Z}_0 = \varnothing$ (i.e., 00000), only the extended child $\{5\}$ (i.e., 00001) exists. For the leaf TEP $\{3, 4, 5\}$ (i.e., 000111), neither extended child nor adjacent child exists. The tree contains exact 26 nodes, enumerating all valid TEPs with $w(\mathbf{e}) \leq 3$. Note that from the root node, the weight-3 TEP 00111 can be reached with 6 steps, compared to $\sum_{i=0}^{2} \binom{5}{i} = 16$ steps in conventional OSD.

To formulate MCTS search on this tree, we define:

- **State** $\mathbf{s}_t = \mathbf{e}_t$: The current TEP $\mathbf{e}_t$ represented by a tree node. Particularly, $\mathbf{s}_0 = \mathbf{0}$ is the root all-zero TEP.
- **Action**: At $\mathbf{s}_t$, action $a_t \in \mathcal{A} = \{\texttt{extend}, \texttt{adjacent}\}$ transitions to the extended or adjacent child TEP.
- **Reward**: Function $R(\mathbf{s}_t)$ assigns large positive rewards to nodes that can lead to the target TEP $\mathbf{e}^*$, while nodes that cannot reach $\mathbf{e}^*$ receive negative rewards.
- **Action value**: $Q(\mathbf{s}_t, a_t)$ denotes the maximum cumulative reward obtainable from state $\mathbf{s}_t$ by taking action a_t.

Remark 1 (Target TEP and codeword)**.** *In practice, the target TEP* $\mathbf{e}^*$ *leading to the transmitted codeword is unknown during decoding. Therefore, for training, we obtain* $\mathbf{e}^*$ *corresponding to the MLD codeword, rather than the actual transmitted codeword. Our policy network learns to find MLD solutions under different noise realizations.*

B. MCTS over TEP Tree

The MCTS search begins with a tree containing only the root node (all-zero TEP), and the tree is incrementally expanded as MCTS performs multiple searches. Each search iteration follows the standard MCTS framework, i.e., steps including *Selection*, *Expansion*, *Simulation* and *Backpropagation*. Figure 2 illustrates an example. A policy neural network guides *Selection*, and is iteratively trained on MCTS search statistics. The training will be explained in Section III-C.

1) Selection: Starting from the root state $\mathbf{s}_0$, we descend the tree by repeatedly selecting the child with the highest PUCT (Predictor and Upper Confidence bounds [23]) score, i.e.,

$$\mathrm{PUCT}(\mathbf{s}_t, a_t) = Q(\mathbf{s}_t, a_t) + c_{\mathrm{puct}}\, p(\mathbf{s}_t, a_t)\, \frac{\sqrt{N(\mathbf{s}_t)}}{1+N(\mathbf{s}_t, a_t)}, \quad (1)$$

where $p(\mathbf{s}_t, a_t)$ is the prior policy from a neural network, $N(\mathbf{s}_t)$ is the visit count for state $\mathbf{s}_t$ and $N(\mathbf{s}_t, a_t)$ is the number of times action a_t has been selected from state $\mathbf{s}_t$. The hyperparameter c_{puct} controls the exploration-exploitation trade-off. At each state $\mathbf{s}_t$, MCTS selects action

$$a_t^* = \arg \max_{a_t \in \mathcal{M}(\mathbf{s}_t)} \mathrm{PUCT}(\mathbf{s}_t, a_t),$$

where $\mathcal{M}(\mathbf{s}_t)$ is the legality mask for valid actions at state $\mathbf{s}_t$. If the resulting child $\mathbf{s}_{t+1} = f(\mathbf{s}_t, a_t^*)$ exists in the tree, descend to it and repeat. Otherwise, proceed to *Expansion*.

2) Expansion: At state $\mathbf{s}_t$, the action a_t^* creates the new child node $\mathbf{s}_{t+1} = f(\mathbf{s}_t, a_t^*)$. We add $\mathbf{s}_{t+1}$ to the tree and initialize $N(\mathbf{s}_{t+1}) = 1$, $N(\mathbf{s}_t, a_t^*) = 0$, and $Q(\mathbf{s}_t, a_t^*) = 0$. We then proceed to *Simulation*.

3) Simulation (Evaluation): Instead of performing simulations as in AlphaZero, we can directly evaluate whether each newly expanded child can reach the target TEP $\mathbf{e}^*$ using the tree structure. Let the new TEP $\mathcal{Z}_{\mathbf{e}_{t+1}} = \{z_1, \ldots, z_\ell\}$ and the target TEP $\mathcal{Z}_{\mathbf{e}^*} = \{z_1^*, \ldots, z_{\ell^*}^*\}$. We note that $\mathcal{Z}_{\mathbf{e}^*}$ can be reached from $\mathcal{Z}_{\mathbf{e}_{t+1}}$ if and only if ① $\ell \leq \ell^*$, ② $z_i = z_i^*$ for $i \in \{1, 2, \ldots, \ell - 1\}$, and ③ $z_\ell > z_\ell^*$. We omit the proof.

For each $\mathbf{s}_{t+1} = \mathbf{e}_{t+1}$, we then obtain the candidate codeword $\mathbf{c}_{\mathbf{e}_{t+1}} = (\mathbf{b}_0 \oplus \mathbf{e}_{t+1})\mathbf{G}$ and assign the reward

$$R(\mathbf{s}_{t+1}) = \begin{cases} +100, & \text{if } \mathcal{Z}_{\mathbf{e}_{t+1}} \text{ can reach } \mathcal{Z}_{\mathbf{e}^*} \\ -d(\mathbf{c}_{\mathbf{e}_{t+1}}, \mathbf{r}), & \text{otherwise,} \end{cases} \quad (2)$$

providing positive reward for paths toward the target TEP.

4) Backpropagation: We propagate the reward of newly expanded child back along the entire path. Let $\mathcal{T} = \{(\mathbf{s}_0, a_0^*), (\mathbf{s}_1, a_1^*), \ldots, (\mathbf{s}_t, a_t^*)\}$ be the trajectory leading to $\mathbf{s}_{t+1}$. For each ancestor state-action pair $(\mathbf{s}_i, a_i^*)$, $i \in \{0, \ldots, t\}$, update $N(\mathbf{s}_i) \leftarrow N(\mathbf{s}_i) + 1$, $N(\mathbf{s}_i, a_i^*) \leftarrow N(\mathbf{s}_i, a_i^*) + 1$, and $Q(\mathbf{s}_i, a_i^*) \leftarrow \max\{Q(\mathbf{s}_i, a_i^*), R(\mathbf{s}_{t+1})\}$.

After backpropagation, one search iteration completes. The process repeats for a maximum of K iterations or until the target TEP $\mathbf{e}^*$ is encountered.

C. Training

For training, a large collection of codeword transmissions is simulated over an AWGN channel to produce the received vectors $\mathbf{r}$ with LLR $\boldsymbol{\ell}$, as described in Section II-A. Each received vector is decoded by the original OSD with a sufficiently large order to obtain the near-MLD codeword $\mathbf{c}_{\mathbf{e}^*}$. The target TEP $\mathbf{e}^*$ is determined by the difference pattern between the first k bits of $\mathbf{c}_{\mathbf{e}^*}$ and the hard-decision vector $\mathbf{b}_0$ derived from $\mathbf{r}$. Let $\mathcal{D}_{\mathrm{train}} = \{(\mathbf{r}, \mathbf{c}_{\mathbf{e}^*}, \mathbf{e}^*)\}$ denote that dataset.

The neural network policy used in (1) is defined as

$$p(\mathbf{s}_t, a_t) = f_\theta\big(\mathbf{e}_t, \mathbf{c}_{\mathbf{e}_t}, d(\mathbf{c}_{\mathbf{e}_t}, \mathbf{r}), \mathbf{G}_{\mathrm{flat}}, \bar{\boldsymbol{\ell}}\big), \quad (3)$$

where $d(\mathbf{c}_{\mathbf{e}_t}, \mathbf{r})$ is the Euclidean distance,$\mathbf{G}_{\text{flat}} \in \{0,1\}^{n\times k}$ is the generator matrix reshaped into a vector by concatenating its rows, and $\bar{\boldsymbol{\ell}} = (\boldsymbol{\ell} - \mu_\ell)/\sigma_\ell$ is the normalized LLR with mean μ_ℓ and standard deviation σ_ℓ computed from $\boldsymbol{\ell}$. The network outputs action probabilities for $a_t \in \mathcal{M}(\mathbf{s}_t)$.

For each training sample $(\mathbf{r}, \mathbf{c}_{\mathbf{e}^*}, \mathbf{e}^*) \in \mathcal{D}_{\text{train}}$, we perform K independent MCTS episodes, each starting from the root state $\mathbf{s}_0 = \mathbf{0}$ following the four-step procedure in in Section III-B. Each episode terminates when the target TEP is found ($\mathbf{e}_t = \mathbf{e}^*$), a step budget M is exhausted, or no legal actions remain. After the K episodes, we compute the visit-count distribution $\pi_t(a_t) = N(\mathbf{s}_t, a_t)/\sum_{a\in M(\mathbf{s}_t)} N(\mathbf{s}_t, a)$ as the training target for each explored state $\mathbf{s}_t$. The state-target pairs $(\mathbf{s}_t, \pi_t)$ are collected in replay buffer $\mathcal{B}$. When $|\mathcal{B}| = B$, we minimize the cross-entropy loss

$$\mathcal{L}_{\text{pol}} = -\frac{1}{B} \sum_{(\mathbf{s}_t,\pi_t)\in\mathcal{B}} \sum_{a_t\in M(\mathbf{s}_t)} \pi_t(a_t) \log p(\mathbf{s}_t, a_t) \tag{4}$$

via stochastic gradient descent (SGD) with learning rate η and Adam optimizer. Then, we clear the buffer. This training procedure follows AlphaZero [18], where MCTS refines the visit-count distributions for the neural network to learn. The training process of a single epoch is summarized in Algorithm 1.

Algorithm 1: Training policy network via MCTS

Input: Dataset $\mathcal{D}_{\text{train}}$, Generator matrix $\mathbf{G}$, max steps M, MCTS episodes K, batch size B, decoding order m
Output: Well-trained policy $f_\theta(\cdot)$

Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$.
foreach $(\mathbf{r}, \mathbf{c}_{\mathbf{e}^*}, \mathbf{e}^*) \in \mathcal{D}_{train}$ **do**
  Initialize $q \leftarrow 0$ and $\mathbf{b_0}$ (hard-decision of first k bits of $\mathbf{r}$).
  for $q = 1, \ldots, K$ **do**
    Initialize the root node of TEP tree $\mathbf{s_0} = \mathbf{e_0} = \mathbf{0}$.
    for $t = 0, \ldots, M$ **do**
      For state $\mathbf{s}_t = \mathbf{e}_t$, obtain $\mathbf{c}_{\mathbf{e}_t}$=$(\mathbf{b}_0 \oplus \mathbf{e}_t)\mathbf{G}$.
      if $\mathbf{e}_t = \mathbf{e}^*$ **then break**
      Select action $a_t^* = \arg\max_{a_t\in\mathcal{M}(\mathbf{s}_t)} \text{PUCT}(\mathbf{s}_t, a_t)$.
      if *no legal action* a_t^* **then break**
      Transit to child state $\mathbf{s}_{t+1}$ by applying a_t^*.
      if *The child* $\mathbf{s}_{t+1}$ *exists* **then**
        continue
      else
        Add $\mathbf{s}_{t+1}$ to the TEP tree.
        Initialize $N(\mathbf{s}_{t+1}) = 1$, $N(\mathbf{s}_t, a_t^*)$=0, $Q(\mathbf{s}_t, a_t^*)$=0.
        Obtain $R(\mathbf{s}_{t+1})$ according to (2).
        Backpropagation as detailed in Section III-B4.
  Obtain $\pi_t(a_t)$ for all visited states, and $(\mathbf{s}_t, \pi_t)$ to $\mathcal{B}$.
  if $|\mathcal{B}| = B$ **then**
    Calculate $\mathcal{L}_{\text{pol}}$ and optimize the policy. Set $\mathcal{B} \leftarrow \varnothing$.

D. Inference (Online Decoding)

During inference, unlike training where the tree is expanded incrementally, the TEP tree is fully pre-generated and stored. The trained policy network guides a depth-first search over the TEP tree without MCTS statistics. Starting from the root state $\mathbf{s}_0$, we iteratively select the action according to:

$$a_t^* = \arg\max_{a_t\in M(\mathbf{s}_t)} p(\mathbf{s}_t, a_t), \tag{5}$$

where $p(\mathbf{s}_t, a_t)$ is the output from trained policy f_θ in (3).

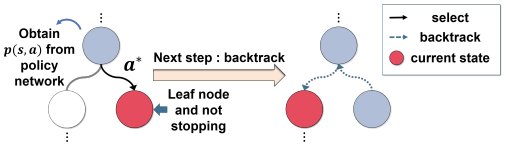


Fig. 3: Example of depth-first TEP search. The search first visits the left child via action a^*, then backtracks to explore the right child.

The MLD results $\mathbf{c}_{\mathbf{e}^*}$ is unknown and must be searched by the decoder. At each visited state $\mathbf{s}_t$, we generate the candidate codeword $\mathbf{c}_{\mathbf{e}_t} = (\hat{\mathbf{b}}_0 \oplus \mathbf{e}_t)\mathbf{G}$ and evaluate it against a stopping criterion. In this paper, we consider either of two criteria:

- *Practical Probability-based Stopping:* We use the stopping condition from PB-OSD [11, Eq. (13)-(14)], which computes the success probability $P_{\text{s}}(\mathbf{c}_{\mathbf{e}_t} \mid \mathbf{d}_t)$ for each codeword $\mathbf{c}_{\mathbf{e}_t}$, where $\mathbf{d}_t$ is the difference pattern between $\mathbf{c}_{\mathbf{e}_t}$ and the hard decision of $\mathbf{r}$. The decoding terminates if $P_s(\mathbf{c}_{\mathbf{e}_t} \mid \mathbf{d}_t) \geq \tau$ for a threshold $\tau \in [0, 1]$, and outputs $\hat{\mathbf{c}} = \mathbf{c}_{\mathbf{e}_t}$ as the decoded result. We apply $\tau = 0.9$.
- *Perfect Stopping:* The search terminates immediately upon finding the MLD codeword $\mathbf{c}_{\mathbf{e}^*}$, which is then output as $\hat{\mathbf{c}}$. This criterion is impractical and only used to evaluate the search efficiency to reach the MLD solution.

We note that the proposed policy-guided search can be used with any other stopping rule from the literature, e.g., [13].

When the stopping criterion is not satisfied at a leaf node, we backtrack to the nearest ancestor with unexplored children and continue with its next highest-probability action (see Fig. 3). If no stopping criterion is met before the step budget M is exhausted, we output the candidate with minimum Euclidean distance, i.e., $\hat{\mathbf{c}} = \arg\min_{\mathbf{c}\in\mathcal{C}_{\text{visited}}} d(\mathbf{c}, \mathbf{r})$, where $\mathcal{C}_{\text{visited}}$ is the set of all visited TEPs' associated codewords. The inference is summarized in Algorithm 2.

Remark 2. *The depth-first strategy with backtracking ensures complete coverage of TEPs up to order m to guarantee BLER if needed, while prioritizing high-probability paths first.*

Algorithm 2: Inference: Policy-Guided Decoder

Input: Received signal $\mathbf{r}$, generator matrix $\mathbf{G}$, decoding order m, full TEP tree, maximum steps M
Output: Decoded results $\hat{\mathbf{c}}$

Initialize $t \leftarrow 0$, $d_{\min} \leftarrow \infty$, and $\mathbf{b_0}$.
for $t = 0, \ldots, M$ **do**
  For state $\mathbf{s}_t = \mathbf{e}_t$, obtain codeword $\mathbf{c}_{\mathbf{e}_t} = (\mathbf{b}_0 \oplus \mathbf{e}_t)\mathbf{G}$.
  if $d(\mathbf{c}_{\mathbf{e}_t}, \mathbf{r}) < d_{\min}$ **then**
    $d_{\min} \leftarrow d(\mathbf{c}_{\mathbf{e}_t}, \mathbf{r})$ and $\hat{\mathbf{c}} = \mathbf{c}_{\mathbf{e}_t}$.
  if $\mathbf{c}_{\mathbf{e}_t}$ satisfies stopping criterion **then return** $\hat{\mathbf{c}}$
  Select action a_t^* according to (5).
  if *No legal action* a_t^* **then**
    Backtrack to most recent ancestor with an unexplored child; set $\mathbf{s}_{t+1}$ to that child.
  else
    Transit to child state $\mathbf{s}_{t+1}$ by applying a_t^*.
return $\hat{\mathbf{c}}$

E. Complexity Consideration

We analyze the decoding complexity of the proposed MCTS decoder and compare it with OSD variants.

1) Neural network complexity.: We intentionally design a lightweight policy network with $h = 3 \sim 6$ hidden layers and 128 hidden units per layer. For input dimension $d_{\text{in}} = k+kn+2n+1$ and two output actions according to (3), one forward pass costs $C_{\text{nn}} = d_{\text{in}} \cdot 128 + (h-1) \cdot 128^2 + 128 \cdot 2 = \mathcal{O}(kn)$ FLOPs, dominated by the $k \times n$ generator matrix. Since most tree nodes have only one legal action, for a tree traversal of N_{MCTS} steps, the number of network calls is approximately $N_{\text{nn}} \approx N_{\text{MCTS}}/3$. For example, Fig. 1 shows that reaching TEP 00111 requires 7 steps but only 2 network calls.

2) TEP search complexity: Both order-m OSD and non-GE OSD require to search $\sum_{j=0}^{m} \binom{k}{j}$ TEPs in the worst case. In contrast, in the proposed MCTS decoder, a well-trained policy network is able to select the shortest path to the target TEP. The worst-case complexity is therefore the maximum tree depth $m(2k-m+1)/2$, which is obtained by repeatedly selecting the adjacent action whenever legal. For instance, with $k = 16$ and $m = 3$, OSD requires 697 TEPs, whereas MCTS examines at most $3 \times (32-3+1)/2 = 45$ nodes.

3) Overall complexity: Let $N_{\text{non-GE}}$, N_{OSD}, and N_{MCTS} denote the average number of TEP evaluations for non-GE OSD, standard OSD, and the proposed MCTS decoder, respectively. Each TEP requires one encoding operation $\mathbf{c_e} = (\hat{\mathbf{b}}_0 \oplus \mathbf{e})\mathbf{G}$ at cost $\mathcal{O}(k(n-k))$. The overall decoding complexities are

$$C_{\text{NonGE-OSD}} = \mathcal{O}(N_{\text{non-GE}} \cdot k(n-k)),$$
$$C_{\text{OSD}} = \mathcal{O}(N_{\text{OSD}} \cdot k(n-k) + C_{\text{GE}}),$$
$$C_{\text{MCTS}} = \mathcal{O}(N_{\text{MCTS}} \cdot k(n-k) + (N_{\text{MCTS}}/3) \cdot kn),$$

where $C_{\text{GE}} = n \cdot \min(k, n-k)^2$ represents the GE overhead in OSD. The proposed decoder achieves $N_{\text{MCTS}} \ll N_{\text{non-GE}}$ and avoids GE entirely compared to OSD.

TABLE I: Hyper-parameters of training

Parameters	**(32,16) eBCH code**	**(48,24) QR code**
order m	5	6
Dataset size $\lvert\mathcal{D}_{\text{train}}\rvert$	1.1×10^5	1.2×10^6
Maximum steps M	70	129
MCTS episodes K	3000	10000
Hidden layers h	3	6
Hidden units	128	
Batch size $\mathcal{B}$	4096	
Learning rate η	10^{-4}	
Epochs	150	
c_{puct} in (1)	1.38	

IV. Experiments and Results

A. Experimental Setup

We evaluate the proposed decoding framework on both a $(32, 16)$ extended BCH (eBCH) code and a $(48, 24)$ quadratic residue (QR) code. For these codes, the non-GE OSD and the proposed MCTS decoder achieve near-MLD performance with order $m = 5$ and $m = 6$, respectively, while the standard OSD achieves it with order $m = 3$. The non-GE OSD and OSD serve as benchmarks. All decoders use the same stopping criterion, allowing a fair comparison of their search efficiency in approaching near-MLD results.

For training, each sample is obtained through simulation with an SNR uniformly sampled from the range $[0, 5]$ dB. The samples are generated to cover as many distinct target TEPs as possible. The main hyper-parameters are summarized in Table I. Note that maximum number M of steps is set to the tree depth $m(2k-m+1)/2$ for efficient training.

During inference, M is set to $\sum_{i=0}^{m} \binom{k}{i}$ to guarantee best BLER. Received signals are generated at each SNR, and policy-guided decoding is performed until 200 block errors are collected. All experiments are conducted on an Ubuntu 24.04 LTS server equipped with an Intel Xeon Platinum 8488C CPU (16 vCPUs, 32 GB RAM).

B. Simulation Results

1) $(32, 16)$ eBCH code: As shown in Figure 4a, for the $(32, 16)$ eBCH code, the proposed decoder achieves near-MLD BLER performance with m=5. Lowering the order m leads to a gradual performance degradation. It achieves the same BLER as non-GE OSD at the same order.

Figure 4b illustrates the average number of TEPs searched under the perfect stopping criterion. Across all orders, the proposed MCTS decoder visits significantly fewer TEPs than the non-GE OSD. For example, at $m = 5$ and SNR 0 dB, non-GE OSD visits over 250 TEPs, while the proposed decoder requires fewer than 20. We highlight that although the TEP count in non-GE OSD increases rapidly with m due to exhaustive enumeration, the growth in MCTS decoding remains moderate. Under the practical probability-based stopping criterion, as shown in Figure 4c, the proposed method still maintains high search efficiency (200 vs 100 vs 700 searches at 3 dB for MCTS, OSD, and non-GE OSD, respectively). It has only a slightly higher count than OSD but without the GE overhead.

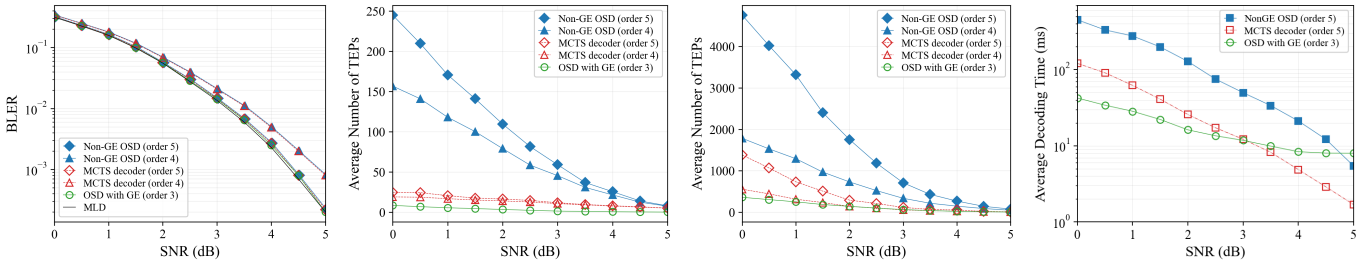
Figure 4d compares the decoding latency of OSD, non-GE OSD, and the proposed MCTS decoding, under the practical stopping. At high SNRs, the runtime of OSD remains nearly constant at around 10 ms, mainly due to the Gaussian elimination step. By removing this step, non-GE OSD achieves a shorter decoding time of about 5 ms. At low SNRs, the MCTS decoder takes longer than OSD due to more search steps, yet it still outperforms OSD at much lower SNRs compared to non-GE OSD.

2) $(48, 24)$ QR code: As shown in Figure 5a, for the $(48, 24)$ QR code, the proposed MCTS decoding achieves near-MLD BLER performance with $m = 6$. Figure 5b and 5c present the average number of visited TEPs under perfect and practical stopping criterion, respectively. The MCTS decoder requires a similar number of searches as OSD, both dramatically fewer than non-GE OSD (e.g., at 5 dB SNR, 31 for MCTS, 10 for OSD, and 1460 for non-GE OSD with practical stopping).

Fig. 5d shows that MCTS decoding is clearly more time-efficient for $(48, 24)$ QR code, with practical stopping criterion. For example, at 5 dB SNR, the average decoding time is 3.5 ms for MCTS, 28 ms for OSD, and 150 ms for non-GE OSD. The delay of OSD saturates at high SNR due to GE overhead, whereas non-GE OSD suffers from excessive search steps leading to higher delays. The decoding time and TEP search counts are summarized in Table II.

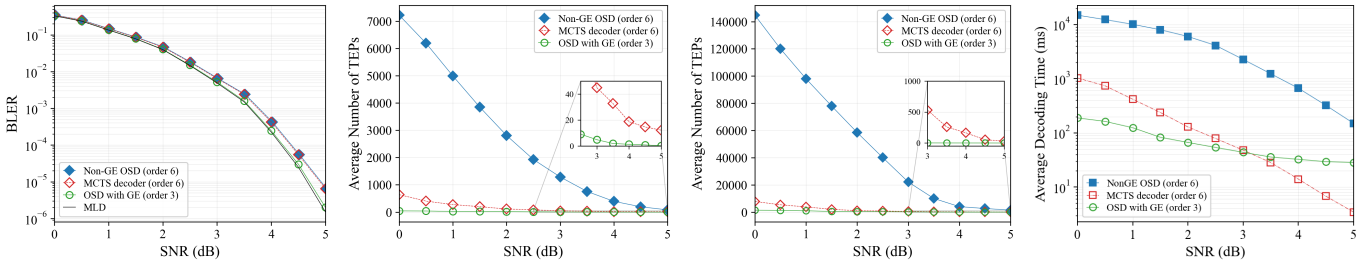
V. Conclusion

In this paper, we proposed policy-guided MCTS decoder for short codes. The proposed decoder attains near-MLD BLER



(a) BLER (b) Avg. TEPs (perfect) (c) Avg. TEPs (practical) (d) Decoding time (practical)

Fig. 4: Comparison of OSD, non-GE OSD and MCTS decoding for (32,16) eBCH code: (a) BLER performance; (b) average searched TEPs under perfect stopping criterion; (c) average searched TEPs under practical stopping criterion; (d) average decoding time under practical stopping criterion.



(a) BLER (b) Avg. TEPs (perfect) (c) Avg. TEPs (practical) (d) Decoding time (practical)

Fig. 5: Comparison of OSD, non-GE OSD and MCTS decoding for (48,24) QR code over AWGN: (a) BLER performance; (b) average searched TEPs under perfect stopping criterion;(c)average searched TEPs under practical stopping criterion; (d) average decoding time under practical stopping criterion.

TABLE II: Decoding time (ms) and average searched TEPs for (48,24) QR code, under the practical stopping criterion.

	1 dB		3 dB		5 dB	
Method	**TEP**	**Time**	**TEP**	**Time**	**TEP**	**Time**
OSD	954.0	124.4	187.0	43.7	10.9	28.4
Non-GE OSD	98016.6	10070.2	22110.0	2271.6	1460.8	150.1
MCTS	3825.7	418.4	532.0	48.0	31.2	3.5

performance while substantially reducing the number of TEPs visited compared to non-GE OSD. At high SNRs, it achieves lower decoding latency than both non-GE OSD earlier and standard OSD. The proposed framework can be seamlessly applied to any existing OSD variants.

REFERENCES

[1] W. Chen, X. Lin, J. Lee, A. Toskala, S. Sun, C. F. Chiasserini, and L. Liu, "5G-advanced toward 6G: Past, present, and future," *IEEE J. Select. Areas Commun.*, vol. 41, no. 6, pp. 1592–1619, 2023.

[2] M. Shirvanimoghaddam *et al.*, "Short block-length codes for ultra-reliable low latency communications," *IEEE Commun. Mag.*, vol. 57, no. 2, pp. 130–137, 2018.

[3] C. Yue, V. Miloslavskaya, M. Shirvanimoghaddam, B. Vucetic, and Y. Li, "Efficient decoders for short block length codes in 6G URLLC," *IEEE Commun. Mag.*, vol. 61, no. 4, pp. 84–90, 2023.

[4] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[5] E. Nachmani and Y. Be'ery, "Neural decoding with optimization of node activations," *IEEE Commun. Lett.*, vol. 26, no. 11, pp. 2527–2531, 2022.

[6] G. Larue, L.-A. Dufrene, Q. Lampin, H. Ghauch, and G. R.-B. Othman, "Neural belief propagation auto-encoder for linear block code design," *IEEE Trans. Commun.*, vol. 70, no. 11, pp. 7250–7264, 2022.

[7] L.-A. Dufrène, Q. Lampin, and G. Larue, "Learning linear block codes with gradient quantization," *arXiv.2503.16169*, 2025.

[8] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theory*, vol. 18, no. 1, pp. 170–182, 2003.

[9] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inform. Theory*, vol. 41, no. 5, pp. 1379–1396, Sep 1995.

[10] C. Yue, C. She, B. Vucetic, and Y. Li, "The guesswork of ordered statistics decoding: Guesswork complexity and decoder design," *IEEE Trans. Inform. Theory*, 2025.

[11] C. Yue, M. Shirvanimoghaddam, B. Vucetic, and Y. Li, "Ordered-statistics decoding with adaptive Gaussian elimination reduction for short codes," in *IEEE GLOBECOM Workshops, GC Wkshps - Proc.* IEEE, 2022, pp. 492–497.

[12] Y. Wu and C. N. Hadjicostis, "Soft-decision decoding of linear block codes using preprocessing and diversification," *IEEE Trans. Inform. Theory*, vol. 53, no. 1, pp. 378–393, 2006.

[13] W. Jin and M. Fossorier, "Probabilistic sufficient conditions on optimality for reliability based decoding of linear block codes," in *IEEE Int. Symp. Inf. Theor. Proc.* IEEE, 2006, pp. 2235–2239.

[14] J. Liang, Y. Wang, S. Cai, and X. Ma, "A low-complexity ordered statistic decoding of short block codes," *IEEE Commun. Lett.*, vol. 27, no. 2, pp. 400–403, 2022.

[15] X. Li, W. Chen, L. Chen, Y. Li, and H. Zhang, "Order skipping ordered statistics decoding and its performance analysis," in *IEEE Inf. Theory Workshop, ITW*. IEEE, 2024, pp. 448–453.

[16] C. Yue, M. Shirvanimoghaddam, Y. Li, and B. Vucetic, "Segmentation-discarding ordered-statistic decoding for linear block codes," in *Proc. - IEEE Glob. Commun. Conf., GLOBECOM*. IEEE, 2019, pp. 1–6.

[17] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[18] ——, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[19] S. Yu, B. Zhang, and Q. Huang, "Ordered statistics derivative decoding for affine-invariant codes without Gaussian elimination," *IEEE Trans. Commun.*, 2025.

[20] M. Fossorier, M. Shakiba-Herfeh, and H. Zhang, "Modified OSD algorithm with reduced Gaussian elimination," *IEEE Commun. Lett.*, vol. 28, no. 8, pp. 1755–1759, 2024.

[21] J. Chen, S. Chen, Y. Qi, and S. Fu, "Intelligent massive MIMO antenna selection using monte carlo tree search," *IEEE Trans. Signal Processing*, vol. 67, no. 20, pp. 5380–5390, 2019.

[22] C. Yue, M. Shirvanimoghaddam, G. Park, O.-S. Park, B. Vucetic, and Y. Li, "Probability-based ordered-statistics decoding for short block codes," *IEEE Commun. Lett.*, vol. 25, no. 6, pp. 1791–1795, 2021.

[23] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.