Convolutional Neural Network-Aided Tree-Based Bit-Flipping Framework for Polar Decoder Using Imitation Learning

Chieh-Fang Teng, Student Member, IEEE, and An-Yeu (Andy) Wu, Fellow, IEEE

Abstract—Known for their capacity-achieving abilities and low complexity for both encoding and decoding, polar codes have been selected as the control channel coding scheme for 5G communications. To satisfy the needs of high throughput and low latency, belief propagation (BP) is chosen as the decoding algorithm. However, it suffers from worse error performance than that of cyclic redundancy check (CRC)-aided successive cancellation list (CA-SCL). Recently, convolutional neural network-aided bit-flipping (CNN-BF) is applied to BP decoding, which can accurately identify the erroneous bits to achieve a better error rate and lower decoding latency than prior critical-set bit-flipping (CS-BF) mechanism. However, successive BF, having better error correction capability, has not been explored in CNN-BF since the more complicated flipping strategy is out of the scope of supervised learning. In this work, by using imitation learning, a convolutional neural network-aided tree-based multiple-bits BF (CNN-Tree-MBF) mechanism is proposed to explore the benefits of multiple-bits BF. With the CRC information as additional input data, the proposed CNN-BF model can further reduce 5 flipping attempts. Besides, a tree-based flipping strategy is proposed to avoid useless flipping attempts caused by wrongly flipped bits. From the simulation results, our approach can outperform CS-BF and reduce flipping attempts by 89% when code length is 64, code rate is 0.5 and SNR is 1 dB. It also achieves a comparable block error rate (BLER) as CA-SCL.

Index Terms—Polar codes, belief propagation, bit-flipping, convolutional neural network, imitation learning.

I. INTRODUCTION

Polar code is a type of block channel code proven to achieve channel capacity [1]. In recent years, it has received intensive attention due to its low complexity for encoding and decoding. In 2016, it was selected by 3GPP as the officially coding scheme for the enhanced mobile broadband (eMBB) control channel of 5G New Radio (NR) [2].

The two main algorithms for polar decoding are successive cancellation (SC) and belief propagation (BP). Compared with BP decoding, SC decoding can fulfill the channel-capacity ability and achieve a lower block error rate (BLER) through enhanced SC algorithms, such as SC list (SCL) [3]–[4] and SC flip (SCF) [5]–[13]. However, SC suffers from high latency and low throughput due to its inherently sequential processing nature, while BP algorithm excels in architectural parallelization, thus has lower decoding latency and better throughput [14]. Recently, considerable efforts have been put into improving the error performance of BP to achieve that of enhanced SC algorithms, while still maintaining its advantages.

In [15]–[17], the BP algorithm is enhanced through the scaling of messages from trainable weights. It can reduce the total number of BP iterations with lower overall complexity, but still does not address the lacking error correction performance. The authors in [18] concatenate the CRC factor graph with the polar factor graph for the exchanging of extrinsic information, which can achieve better error correction performance. A BP list (BPL) decoder is proposed in [19]–[21] to achieve comparable performance as SCL by performing BP algorithm on different permuted factor graphs or permuted codewords. In [22], post-processing is applied to different types of BP errors by performing enhancement and perturbation on the reliable and unstable bits, respectively. A similar concept was proposed in [23] by iteratively finding the stable bits and strengthening them as “frozen bits”. However, even with these enhanced approaches, all of them still can not achieve comparable performance as cyclic redundancy check (CRC)-aided SCL (CA-SCL) [3]–[4].

Recently, bit-flipping (BF) decoders also attract a lot of attention because it can minimize the effect of error propagation by performing error corrections on incorrectly decoded bits during decoding iterations [5]–[13], [24]–[29]. In [5]–[13], BF has been successfully applied to SC, but it further deteriorates the decoding latency. On the other hand, with the aid of BF, the BLER of BP-based decoders can be comparable with that of CA-SCL at low to medium SNRs [24]–[25]. Moreover, BP flip (BPF) decoders are realized in [27]–[28] to demonstrate its great throughput and comparable performance as SCL. Recently, as the fast-emerging field of deep learning-assisted communication systems, many researchers start exploiting the benefit of deep learning for the design of channel coding [30]–[32]. A convolutional neural network-aided BF (CNN-BF) was proposed in [29], which can dynamically identify the erroneous bits from the metadata of BP decoding as shown in Fig. 1(a). Thus, it can achieve better error correction capability and lower decoding.
Early termination mechanism can be generated by multiplying $C(1 - K)$. The most $nB$ polarizing transformation information bits are first assigned to the $\log_2 N$ times to exploit the channel polarization and $D$ Design of tree-based undo model. Design of convolutional neural network-aided bit-flipping by learning. We propose a convolutional neural network-aided tree-based bit-flipping framework for polar decoder: (a) 1-bit bit-flipping [29], and (b) proposed multiple-bits bit-flipping.

In this paper, by taking advantage of imitation learning, we propose a CNN-aided bit-flipping for polar decoder: (a) 1-bit bit-flipping [29], and (b) proposed multiple-bits bit-flipping. However, some worse codewords that demand successive flipping attempts cannot be successfully decoded in [29]. To further explore the benefit of BF and the great potential of deep learning, we extend [29] from 1-bit BF to multiple-bits BF as shown in Fig. 1(b). However, two critical issues should be addressed:

1) Early termination mechanism: For the BF mechanism, it sets the a priori knowledge of information bit to infinity to avoid error propagation. However, for multiple-bits BF, suppose the last flipped bit is incorrectly set as a frozen bit, the following flipping attempts are useless. Thus, it requires an effective mechanism to determine whether to continuously flip the next bit or undo the wrongly flipped bit.

2) Deficiency of supervised learning: For 1-bit BF strategy, if the flipped bit cannot result in correct decoding, it always undoes the action and sequentially attempts other candidates. Thus, the subsequent flipping is independent with the previous attempts and supervised learning can be easily applied for the training. However, as the mechanism extended to multiple-bits BF, the next flipping action is based on the current decoding state and flipping action, which means that the training data cannot be easily obtained and is out of the scope of supervised learning.

In this paper, by taking advantage of imitation learning, we propose a convolutional neural network-aided tree-based multiple-bits BF (CNN-Tree-MBF) mechanism, which can significantly improve the error correction capability and reduce decoding latency by exploring the benefit of multiple-bits BF. Our main contributions are summarized as below:

1) Design of convolutional neural network-aided bit-flipping model: To further enhance the prediction accuracy of the CNN-BF model, we transform the CRC results to an image which means that the training data cannot be easily obtained. However, as the mechanism extended to multiple-bits BF, the next flipping action is based on the current decoding state and flipping action, which means that the training data cannot be easily obtained and is out of the scope of supervised learning.

In this paper, by taking advantage of imitation learning, we propose a convolutional neural network-aided tree-based multiple-bits BF (CNN-Tree-MBF) mechanism, which can significantly improve the error correction capability and reduce decoding latency by exploring the benefit of multiple-bits BF. Our main contributions are summarized as below:

1) Design of convolutional neural network-aided bit-flipping model: To further enhance the prediction accuracy of the CNN-BF model, we transform the CRC results to an image which means that the training data cannot be easily obtained. However, as the mechanism extended to multiple-bits BF, the next flipping action is based on the current decoding state and flipping action, which means that the training data cannot be easily obtained and is out of the scope of supervised learning.

2) Design of tree-based undo model: A tree-based Undo model for flipping strategy is proposed to strike a good balance between the benefit of multiple-bits BF and the increased flipping attempts caused by wrongly flipped bits. Besides, a CNN-aided Undo model is also proposed, which can be combined into the tree-based Undo model to further reduce flipping attempts by 19%.

3) Multiple-bits bit-flipping mechanism using imitation learning: A multiple-bits BF mechanism, composed of the aforementioned BF model and the Undo model, is proposed as shown in Fig. 1(b). By iteratively alternating between the stages of training and data generation, these two models can be well-trained by imitation learning. From the simulation results, it can outperform state-of-the-art critical set bit-flipping (CS-BF) algorithm [24] with 89% reduction in flipping attempts. Besides, by increasing the maximum number of flipping attempts, it can achieve comparable performance as CA-SCL with slightly increased average decoding latency.

The rest of this paper is organized as follows. Section II briefly reviews BP decoding and prior BF works. In Section III, after detailed analyses of the benefits and challenges of multiple-bits BF, we propose our multiple-bits BF mechanism. Based on this mechanism, the design of the CNN-BF model and the tree-based Undo model are shown in Section IV. The numerical experiments and complete analyses are shown in Section V. Finally, Section VI concludes our work.

II. POLAR CODES AND DECODING ALGORITHMS

A. Polar Codes

To construct an $(N, K)$ polar code, the $N$-bits message $u$ is recursively constructed from a $2 \times 2$ polarizing transformation $\lambda = [\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}]$ by $\log_2 N$ times to exploit the channel polarization [1]. As $N \to \infty$, the synthesized channels tend to two extremes: the noisy channels (unreliable) and noiseless channels (reliable). Therefore, the $K$ information bits are first assigned to the $K$ most reliable bits in $u$ and the remaining $(N - K)$ bits are referred to as frozen bits with the assignment of zeros. Besides, the $r$-bits cyclic redundancy check (CRC) is attached as the information bits, which can be utilized to check the correctness of decoded results. Thus, the actual information rate is $(K - r)/N$. Then, the $N$-bits transmitted codeword $x$ can be generated by multiplying $u$ with generator matrix $G$ as follow:

$$x = Gu = F^{\otimes n} Bu, \quad n = \log_2 N.$$  \hspace{1cm} (1)

$F^{\otimes n}$ is the $n$-th Kronecker power of $F$ and $B$ represents the bit-reversal permutation matrix.

B. Belief Propagation Decoding Algorithm

Belief propagation is a widely used message-passing algorithm for decoding, such as in low-density parity-check (LDPC) codes and polar codes. The decoding process of polar codes is to iteratively apply BP algorithm over the corresponding factor graph as shown in Fig. 2. For an $(N, K)$ polar code, there are $n = \log_2 N$ stages and a total of $N \times (n + 1)$ nodes on the factor graph. Each node $(i, j)$ represents the $j$-th node at the $i$-th stage in the factor graph. It has two types of log likelihood ratios (LLRs), namely left-to-right message $R_{i,j}^{(l)}$ and
right-to-left message $L_{i,j}^{(t)}$, where $t$ represents the $t$-th iteration. Before beginning the iterative propagation and the updating of node values, their LLR values are first initialized as:

$$R_{1,j}^{(1)} = \begin{cases} 0, & \text{if } j \in A \\ +\infty, & \text{if } j \in A^c \end{cases},$$

where $A$ and $A^c$ are the set of information bits (including CRC bits) and the set of frozen bits, respectively.

Then, the iterative decoding procedure with the updating of $R_{i,j}^{(t)}$ and $L_{i,j}^{(t)}$ is given by:

$$
\begin{align*}
R_{1,j}^{(1)} &= \ln \frac{P(y_j | x_j = 0)}{P(y_j | x_j = 1)}, \\
L_{i,j}^{(t)} &= \begin{cases} g(L_{i+1,j}^{(t-1)} + R_{i,j}^{(t-1)}), & \text{if } j \in A^c \\ g(R_{i,j}^{(t-1)}), & \text{if } j \in A \\ L_{i,j}^{(t-1)}, & \text{if } j \in A^c, \\ \end{cases} \\
R_{i+1,j}^{(t+1)} &= \begin{cases} g(L_{i+1,j}^{(t)} + R_{i,j}^{(t)}), & \text{if } j \in A^c \\ g(R_{i,j}^{(t)}), & \text{if } j \in A \\ R_{i,j}^{(t)}, & \text{if } j \in A^c, \\ \end{cases} \\
R_{i,j}^{(t+1)} &= \begin{cases} g(L_{i,j+1}^{(t)} + R_{i,j}^{(t)}), & \text{if } j \in A^c \\ g(R_{i,j}^{(t)}), & \text{if } j \in A \\ R_{i,j}^{(t)}, & \text{if } j \in A^c, \\ \end{cases}
\end{align*}
$$

where $g(a, b) \approx \text{sign}(a) \text{sign}(b) \min(|a|, |b|)$ is the min-sum approximation introduced to reduce complexity. Finally, after $T$ iterations, the estimation of $\hat{u}$ is decided by:

$$\hat{u}_j = \begin{cases} 0, & \text{if } L_{i,j}^{(T)} + R_{i,j}^{(T)} \geq 0, \\ 1, & \text{if } L_{i,j}^{(T)} + R_{i,j}^{(T)} < 0. \\ \end{cases}$$

With the CRC, the decoded results can be checked by multiplying with the CRC parity check matrix $H_{CRC}$ as follows:

$$c = H_{CRC} \hat{u}.$$

If $c$ is a zero vector, it means that the decoded results pass CRC; otherwise, the codeword fails to decode successfully.

**C. Prior Work: Belief Propagation Bit-Flipping Decoding Algorithms [24]–[29]**

Due to the message passing algorithm of BP decoding, the incorrectly estimated information bits may result in error propagation and thus negatively affect the reliability of many other bits. To address the issue of error propagation, BF is an assistive mechanism to the decoding process, where a possibly incorrectly decoded bit is guessed and flipped prior to the restarted decoding process. Thus, precise BF can effectively improve the block error rate (BLER) performance for polar codes. The mechanism of BF flips the value of previous estimated $\hat{u}_j$, and sets the a priori knowledge of $u_j$ to infinity as if it is a frozen bit. Therefore, the initialized values of $R_3$ in (2) are revised as:

$$R_{1,j}^{(1)} = \begin{cases} 0, & \text{if } j \in \{A \setminus F\} \\ +\infty, & \text{if } j \in F \\ \end{cases}$$

where $F$ is the set of current flipped bits and thus the bits in $F$ will be set to infinity. $d_j \in \{-1, 1\}$ represents the flipping direction. If $d_j = 1$, it means to flip the bit $\hat{u}_j$ to the opposite direction of $L_{1,j}^{(T)} + R_{1,j}^{(T)}$. Vice versa, it strengthens the bit $\hat{u}_j$ to the same direction. By doing so, the a priori knowledge of flipped bits is expected to correct the other wrongly propagated messages in the previously failed BP decoding.

According to the algorithm detailed above, the decoding latency of BF corresponds to the required number of flipping attempts, which is dominated by the correction of flipped bits. Therefore, critical set (CS), consisting most of the error-prone bits, was proposed and adopted in [7]–[8], [10]–[11], [24]–[25]. The critical set is constructed based on the structure of polar code, where the first nodes in the subtrees of all information bits are at high risk, and thus are included in the set. By only selecting bits for BF from CS, it results in less flipping attempts and achieves lower latency. Furthermore, CS with order $\omega$ is proposed in [24] to flip $\omega$ error-prone bits simultaneously, which has better error correction capability at the cost of 2$^\omega$ times increase in flipping attempts. In [25], error types for BF are analyzed and multiple bit-flipping sets are dynamically generated based on the submatrix check, which can remove unnecessary flipping positions and increase the order of CS.

Though critical set can effectively reduce the search space for flipping attempt, this mechanism is essentially still a process of trial-and-error to attempt all the bits in critical set. Besides, it also excludes error bits outside the critical set, thus degrading the error correction capability of BF. In [27], BP flip decoder is realized with efficient identification of error-prone bits, which is based on the magnitude of LLR values and 3GPP standard. Furthermore, [27] is extended in [28] by presenting a hardware-friendly high-order BF mechanism. Recently, a convolutional neural network-aided bit-flipping (CNN-BF) was proposed in [29] to exploit the variation of BP decoding process and dynamically identifies the erroneous bit for 1-bit correctable codewords, which can achieve better decoding performance and lower flipping attempts compared to [24].

After the flipping order is established from CS [24] or predicted by CNN model [29], the conventional BP decoding process can commence. If BP fails to decode successfully, checked by a cyclic redundancy check (CRC), a candidate bit from the flipping order is sequentially selected for flipping according to Eq. (6) as shown in Fig. 1(a). After BF, BP decoding is performed again. If the result satisfies CRC, the decoding process is completed. Otherwise, it undoes the last flipped bit and attempts the other candidates in the flipping order. The process is iteratively performed until CRC is successfully passed. For more details about BF and critical set, please refer to [5]–[13], [24]–[29].
In Fig. 3, the performance of exhaustive BF can be seen as the optimal performance due to the greatest searching space includes all combinations. However, the performance of opposite BF is only slightly lower than that of exhaustive BF, which means that flipping the bits to the opposite direction is a more efficient strategy to achieve good performance at a reasonable number of flipping attempts. On the other hand, for the performance of CS-based mechanisms, we can observe that the improvement by increasing \( \omega \) is limited, especially for opposite CS-BF. This is due to the constrained searching space and it is hard to simultaneously flip multiple bits to the correct directions without exhaustive searching.

In [29], the authors adopted the mechanism of opposite 1-bit BF and dedicated to reducing the time complexity from \( O(I \times P^K) \) to \( O(I) \) by improving the accuracy of the flipping order. However, from Fig. 3, we can observe that there are about 20% error codewords can not be corrected by opposite 1-bit BF. Therefore, we want to extend [29] from 1-bit BF to multiple-bits BF in this work. By doing so, the error correction capability of BF can be greatly improved and the superiority over CS-BF will also be more apparent.

**B. Challenges for Neural Network-Aided Multiple-Bits Bit-Flipping**

For the training of the neural network-aided 1-bit BF model, it can be easily divided into two steps. Firstly, collecting all the received codewords, which fail to decode successfully after the first BP decoding. Secondly, we need to label the correct flipping position for these codewords. The process exhaustively goes through the \( K \) information bits by setting the a priori knowledge of \( \mathbf{u} \) to opposite infinity as a frozen bit and followed by BP decoding. If the newly decoded result is correct, the label \( \mathbf{b} \) is set to 1; vice versa, it is labeled to 0. In this way, the labeled data for these failed decoded codewords can be obtained and the neural network can be trained using supervised learning.

However, the above flow is only applicable to 1-bit BF model. Because if the decoded result still fails to pass CRC, it will undo the last flipped bit and attempt other candidates in the flipping order. Thus, the subsequent flipping is independent with the previous attempts. However, as the mechanism extended to multiple-bits BF, the next flipping action is based on the current decoding state and current flipping action, which is out of the scope of supervised learning. For example, the next flipped position can be classified into three categories:

- **Correct position**: after flipping this position, the next BP decoding will successfully decode the codeword, which is labeled to 1.
- **Wrong position**: the flipped position will cause unrecoverable error bit (e.g., \( u_j \) is 1, but \( R_{i,j}^{(1)} \) is set to \( +\infty \)), which is labeled to 0.
- **Intermediate position**: the flipped position will not cause unrecoverable error bit (e.g., \( u_j \) is 0 and \( R_{i,j}^{(1)} \) is set to \( +\infty \)), but the next BP decoding still fails to decode the codeword successfully, which is also labeled to 0.

---

**III. PROPOSED MULTIPLE-BITS BIT-FLIPPING MECHANISM USING IMITATION LEARNING**

**A. Analysis of Different Bit-Flipping Mechanisms**

Before introducing our proposed multiple-bits BF, we first evaluate the error correction capability and time complexity between four different BF mechanisms as shown in Fig. 3. In Fig. 3, the analysis of CS-BF is based on [24], which means that only the bits in CS are considered for flipping. Therefore, the time complexity is proportional to the size of the critical set \(|\text{CS}|\). Besides, \( \omega \) denotes the number of simultaneously flipped bits. On the other hand, without constrained by CS, it can flip arbitrary bits in the set of information bits. Furthermore, another difference is that it adopts a progressive strategy, which means it progressively increases the number of flipped bits to \( \omega \) by flipping one bit at a time. For example, suppose \( \omega = 2 \), it will firstly flip one bit followed by BP decoding. If it still fails to decode successfully, it continuously flips another bit and applies BP decoding again. This strategy is more consistent with our scenario to apply neural networks to predict the error probability for each bit and only flip the bit with the highest error probability. Therefore, the time complexity for this progressive strategy is proportional to \( 2^\omega \) instead of \( C^K_{\omega} \), where \( P \) and \( C \) represent permutation and combination, respectively.

The term of “exhaustive” means to flip the bits in both directions, namely with \( d_j \in \{-1, 1\} \). Thus, it has \( 2^\omega \) different combinations for flipping directions and the time complexity is proportional to \( 2^\omega \). Vice versa, the term of “opposite” means that it only attempts the opposite direction with \( d_j = 1 \), which can effectively reduce the flipping attempts at the cost of degraded error correction capability. However, its time complexity is reduced to only proportional to \( \omega \). In Fig. 3, the maximum number of flipped bits \( \omega \) is set to 3 and \( I \) denotes the number of iterations for each BP decoding. Because each BF strategy will attempt all flipping combinations until it correctly decodes the codeword, we can obtain the best performance that each strategy can achieve. Besides, the error correction capability of smaller \( \omega \) is included in bigger \( \omega \). Thus, we use the metric of cumulative correct percentage to evaluate the improved error correction capability as the increase of \( \omega \).

In [29], the authors adopted the mechanism of opposite 1-bit BF and dedicated to reducing the time complexity from \( O(I \times P^K) \) to \( O(I) \) by improving the accuracy of the flipping order. However, from Fig. 3, we can observe that there are about 20% error codewords can not be corrected by opposite 1-bit BF. Therefore, we want to extend [29] from 1-bit BF to multiple-bits BF in this work. By doing so, the error correction capability of BF can be greatly improved and the superiority over CS-BF will also be more apparent.
Therefore, the next action is based on the current state and current action. Suppose the current action flips the wrong position, the next action must undo to escape from the unrecoverable state. Besides, the different intermediate positions will also result in different correct positions for the next action. This challenge was also firstly pointed out in [12], where the authors adopted a long short-term memory (LSTM) network to continuously predict the first error bit for SC decoding. Similarly, the previous action will also affect the following action in that scheme.

In conclusion, multiple-bits BF will correlate the previous actions and the subsequent actions and thus is out of the scope of supervised learning. The problem of multiple-bits BF is just like a multi-step maze game or a self-driving car problem, where multiple decisions are made for a success to the destination. The mission of our designed neural network model is to find out the correct action (flipping position) based on the state (designed input data) in each step. Consequently, to address this challenge, we need to exploit imitation learning, which has been successfully applied to the training of self-driving cars [33].

C. Multiple-Bits Bit-Flipping Using Imitation Learning

According to the previous discussions, we need two kinds of models, namely the BF model and the Undo model, to accomplish the multiple-bits BF mechanism. The BF model is used to predict the error probability for each bit and the Undo model is utilized to indicate the correctness of last BF action and whether an undo action is required to rescue the wrongly flipped bit. Therefore, it is difficult to train these two models using supervised learning due to the action of these two models also affect the following action in that scheme.

In this part, if the last flipped bit belongs to the wrong position, the label $m$ is set to 1; vice versa, the position must belong to the intermediate position and thus be labeled to 0. Then, the label $m$ and BP’s metadata are also collected into the Undo data pool. Finally, the Undo model predicts the error probability of the last flipped position. If the probability is greater than 0.5, it will undo the last flipped bit and attempt the next bit in the flipping order. On the other hand, it will enter the BF model again for the training stage will transit from inference to the training stage as shown in Fig. 4(a). The BF model and the Undo model will be optimized based on the training data in the data pool. Because the model’s behavior will change after training, the training data in the data pool must also be updated. Thus, the process will transit to the inference stage again. By iteratively using imitation learning for the process of training and data generation, these two models can converge and learn to accomplish the multiple-bits BF mechanism. This iterative process will continue until the accuracy on the validation set cannot be further improved, which means that both models have converged to a steady point. The detailed algorithm flow will be provided in the next section.

1) Bit-flipping prediction: Firstly, the received codewords will be decoded by BP algorithm. Then, the failed decoded codewords will be sent to the BF model. In this state, if generating training data is required, the BP’s metadata and label $b$, which can be obtained according to Section III.B, will be jointly collected into the BF data pool as the training data pair. Then, the BF model predicts the error probability for each bit to construct the flipping order.

2) Belief propagation decoding: Next, the BP decoding will be executed again after flipping the bit with the highest error probability. Then, the failed decoded codewords will enter into the Undo model.

3) Undo prediction: In this state, if generating training data is enabled, the process of labeling and collecting BP’s metadata will be performed. However, the labeling process is slightly different from the process for the BF model. In this part, if the last flipped bit belongs to the wrong position, the label $m$ is set to 1; vice versa, the position must belong to the intermediate position and thus be labeled to 0. Then, the label $m$ and BP’s metadata are also collected into the Undo data pool. Finally, the Undo model predicts the error probability of the last flipped position. If the probability is greater than 0.5, it will undo the last flipped bit and attempt the next bit in the flipping order. The above process will be iterated until enough training data is generated or the stopping criteria for multiple-bits BF mechanism are met. After collecting enough training data, the stage will transit from inference to the training stage as shown in Fig. 4(a). The BF model and the Undo model will be optimized based on the training data in the data pool. Because the model’s behavior will change after training, the training data in the data pool must also be updated. Thus, the process will transit to the inference stage again. By iteratively using imitation learning for the process of training and data generation, these two models can converge and learn to accomplish the multiple-bits BF mechanism. This iterative process will continue until the accuracy on the validation set cannot be further improved, which means that both models have converged to a steady point. The detailed algorithm flow will be provided in the next section.
IV. PROPOSED BIT-FLIPPING MODEL AND UNDO MODEL DESIGNS

After completing the design of the multiple-bits BF mechanism shown in Fig. 4(b), the remaining question is how to design the BF model and the Undo model, which has a great impact on the error correction capability and the required number of flipping attempts. The better prediction accuracy of the BF model can accurately identify the erroneous bits and thus improve the error correction capability and reduce the flipping attempts. On the other hand, the Undo model can do early termination to avoid useless flipping attempts and rescue the codewords from an unrecoverable state. In the following, we will describe the design of these two models in detail.

A. Convolutional Neural Network-Aided Bit-Flipping Model

Firstly, we focus on the design of the BF model. The input data for the model-based approach is important since it has a significant impact on prediction accuracy. Compared with [29], we not only make use of the metadata from BP, but further take advantage of the CRC results $c$ as the input data for the training and prediction.

In each BP decoding iteration, the values of LLRs $\mathbf{L}(t)$, $\mathbf{R}(t)$ on the whole factor graph are recorded and mapped to an image as shown in Fig. 5(a). Besides, the decoded results $\hat{u}(t)$ and CRC results $c(t)$ for each BP iteration can be obtained according to Eq. (4) and Eq. (5), respectively. The values $\mathbf{L}(t)$, $\mathbf{R}(t)$, and $c(t)$ are preprocessed and jointly used as the input data for the following neural network model. Though CRC results $c(t)$ can only be utilized for error-detection, it may potentially indicate which information bits are possibly erroneous because each bit in $c(t)$ is connected with part of the information bits. Suppose $c(t)$ is 1, it means that at least one of its connected information bits is wrongly decoded. Thus, it is informative for neural networks to further analyze and dynamically identify the most likely error bits with greatly improved prediction accuracy, which will be demonstrated in Section V. Also, due to the iterative decoding process, the images and CRC results, representing different iterations, will be jointly integrated as the input data. Therefore, the adopted model can explore not only the relation between connected nodes but also the variation of LLRs and CRC results among different iterations, namely in both spatial and temporal dimensions, which further enhance the prediction accuracy.

In addition, to further improve the prediction accuracy and reduce model complexity, we apply some domain-specific signal pre-processing for the LLR values before feeding the input data into the model. Two features, the absolute and sign values, are extracted from LLRs as shown in Fig. 5(a). By doing so, the absolute values represent the reliability of each node and the sign values are helpful for the model to further explore the variation between different nodes. Besides, since the values of $\text{abs}(|\mathbf{R}|)$ are around either 0 or $\infty$, which are not suitable for the concatenated model to extract features. Thus, the images of $\text{abs}(|\mathbf{R}|)$ are removed without affecting the prediction accuracy but reducing the model complexity. Suppose that the number of iterations for BP is 5, there will be a total 15 images after data pre-processing with each image resolution being $(n + 1) \times N$, which is consistent with the size of the factor graph. The CRC results in each BP iteration are also integrated as an image with the resolution being $r \times 5$ as shown in Fig. 5(a), which is very suitable for neural networks to analyze the variation in the temporal domain and figure out error-prone bits.

Two convolutional neural networks (CNNs) are employed to deal with the image-based input data $\mathbf{I}_1$ and $\mathbf{I}_2$, which correspond to the preprocessed factor graphs and CRC results, respectively. CNN is the widely used model for image processing with the ability to extract local connectivity and subtle features of the input images. After feature extraction, these two CNNs will be concatenated to jointly predict the error probability as shown in Fig. 5(b). Because our problem is not as complicated as in computer vision, it is unnecessary to construct a massive CNN model, such as AlexNet or ResNet. Thus, two tiny CNN models with similar architecture are proposed as shown in Fig. 5(b). For each CNN model, it is mainly constructed by three two-dimensional...
Algorithm 1: Proposed CNN-Aided Tree-Based Multiple-Bits Bit-Flipping (CNN-Tree-MBF) Decoder.

Input: $y, A, d, w, T_{\text{max}}, H_{\text{CRC}}, (u)$

Output: $\hat{u}$

1: $L, R \leftarrow$ initialize the BP decoder using (2)
2: $\hat{u}, L, R \leftarrow$ BP decoder($L, R$)
3: $t \leftarrow 0$
4: $l \leftarrow 1$
5: if $\hat{u}$ does not pass CRC and $t < T_{\text{max}}$ then
6: $\hat{u}_{\text{old}} \leftarrow \hat{u}$
7: Multiple-Bits BF Subtree($L, R, \hat{u}_{\text{old}}, t, l, (u)$)
8: else
9: return $\hat{u}$
10: end if

conventional layers for high-level feature extraction and one dense layer for feature transformation. After concatenating the extracted features from these two CNN models, three dense layers are followed for classification based on the nonlinear combinations of these extracted features from CNN models.

The values below the convolutional layer represent the number and size of filters, respectively. On the other hand, the values below the dense layer represent the number of nodes. Because of the image resolution of $I_2$ is smaller than $I_1$, we set a smaller number of nodes for the dense layer of $I_2$ to reduce the number of parameters. Besides, a batch normalization layer is applied to the input images to normalize the input data and improve the convergence speed. Two maxpooling layers are also used to reduce the dimensionality of the feature map for the reduction of computational complexity and memory overhead. The nonlinear activation function, Rectified Linear Units (ReLUs), among each layer is defined as:

$$f_{\text{ReLU}}(x) = \max\{0, x\}. \quad (7)$$

It is helpful for extracting more complex features. Besides, to reduce overfitting, the regularization technique of “dropout” that avoids updating the weights of part nodes, is also utilized to improve the inference accuracy.

For the problem of BF prediction, the output layer has $K$ nodes, which represents the probability of each bit being flipped or not. The labeled data for training is a vector with $K$ values being 0 or 1 to indicate which bits could be flipped to result in successful decoding as shown in Fig. 5(c). Note that for some input cases, there could be more than 1 position to result in successful decoding. Consequently, this is a multi-label classification problem and the output must be rescaled into the range $[0, 1]$ with sigmoid function to indicate the probability as below:

$$f_{\text{Sigmoid}}(x) = \sigma(x) = (1 + e^{-x})^{-1}. \quad (8)$$

Also, the loss function is cross-entropy, defined as:

$$L(b, \hat{b}) = - \frac{1}{K} \sum_{i=0}^{K-1} b_i \log(\hat{b}_i) + (1 - b_i) \log(1 - \hat{b}_i), \quad (9)$$

where $b_i$ and $\hat{b}_i$ denote the labeled data and predicted value for the $i$-th output, respectively.

B. Tree-Based Undo Model With Preorder Traversal

The Undo model is equally important as the BF model because the wrongly flipped bits can only be corrected by the Undo model. Suppose the Undo model can not effectively undo the wrongly flipped bits, the following flipping attempts are useless. On the other hand, suppose the last flipping attempt flips an intermediate position, the flipping process must be continued or it will degrade the correction capability. The previous flipping position and the action adopted by the Undo model can lead to four different cases:

- True positive (TP): the last flipping attempt flips a wrong position and the Undo model correctly predicts “Undo”. This will avoid useless flipping attempts.
- True negative (TN): the last flipping attempt flips an intermediate position and the Undo model correctly predicts “No Undo”. This will not degrade the error correction capability.
- False positive (FP): the last flipping attempt flips an intermediate position, but the Undo model wrongly predicts “Undo”. This will degrade the error correction capability.
- False negative (FN): the last flipping attempt flips a wrong position, but the Undo model wrongly predicts “No Undo”. The following flipping attempts will be useless.

Thus, the Undo model must strike a good balance to take advantage of the error correction capability from the multiple-bits BF mechanism while avoiding a significantly increased number of flipping attempts caused by wrongly flipped bits.

To satisfy the aforementioned requirements, we firstly propose a criteria-based Undo model, where the actions determined by the Undo model are based on a tree structure as shown in Fig. 6(a). The tree structure can be determined by two kinds of parameters. One is the maximum depth $d$, which represents the maximum number of successively flipped bits for each codeword and is equivalent to $\omega$. Another one is the number of children $w_i$ for the nodes in level $i$, representing $w_i$ bits with the highest error probability will be attempted for each BF prediction. After determining the tree structure, the actions adopted by the Undo model are completely based on the tree’s preorder traversal, which is a depth-first search to better explore the benefit of multiple-bits BF. Our approach has a similar concept as [6] and [28], but both of them are based on a breadth-first search. The “Undo” action is performed only when the multiple-bits BF mechanism traverses to the maximum depth; otherwise, it successively flips other bits based on a new BF prediction.

Take Fig. 6(a) as an example. The received codeword is first decoded by using BP algorithm. Suppose it fails to decode successfully, the BF model predicts the error probability as indicated by the red node and $w_1$ bits with the highest error probability are kept in descending order. Based on the order, the position with index 6 is chosen for flipping and followed by BP decoding. Because it has not reached the maximum depth, the BF model predicts the error probability again based on the updated $L, R$, and $c$. Then, the position with index 4 is flipped and followed by BP decoding. Now, because it has reached the maximum depth, the “Undo” action is performed. So, it returns to the red node and continuously attempts other $w_2 - 1$ bits. Suppose it still fails, the “Undo” action is performed again and it returns to the level 1.
Then, the position with index 4 is chosen and followed by a similar flow. Each row in the flipping order is corresponding to the set $F$ in Eq. (6). The process is performed recursively until the codeword passes CRC or it completely traverses the entire tree or reaches the maximum traversal attempts $T_{\text{max}}$.

Based on the tree-based Undo model and the adjustment of the tree’s maximum depth, we can exploit the benefit of multiple-bits BF without significantly increasing the number of flipping attempts. Besides, it is worth noting that [29] can be seen as a special case of this tree structure when the maximum depth $d$ is set to 1.

### C. Tree-Based Convolutional Neural Network-Aided Undo Model With Preorder Traversal

According to the above discussions, the proposed tree-based Undo model has a deficiency that the “Undo” action will be performed only when reaching the maximum depth. Suppose the wrongly flipped bit occurs in level 1, the subsequent flipping attempts on that subtree are all useless and thus result in longer decoding latency. To address this issue, we propose a tree-based CNN-aided Undo model based on the tree-based Undo model with a small modification as shown in Fig. 6(b). We can observe a blue node, representing a CNN-aided Undo model, is inserted before entering into the BF prediction in level 2. By taking advantage of this model, the wrongly flipped bit can be detected and early “Undo” action can be adopted to avoid the following useless flipping attempts. Suppose the position with index 6 belongs to the wrong position and the CNN-aided Undo model successfully identifies this bit, the following flipping attempts on the subtree can be skipped to reduce the decoding latency.

**Algorithm 2:** Multiple-Bits BF Subtree().

**Input:** $L, R, \hat{u}_{old}, t, l, (u)$

**Output:** $\hat{u}$

1. $I_1, I_2, c \leftarrow$ input data pre-processing$(L, R, H_{\text{CRC}})$
2. if generate training data then
3. $b \leftarrow$ label BF training data$(L, R, u)$
4. BF data pool $\leftarrow (I_1, I_2, c, b)$
5. end if
6. $\hat{b} \leftarrow$ CNN-aided BF model$(I_1, I_2, c)$
7. for $j = 1 : w$ do
8. $L, R \leftarrow$ initialize the BP decoder using (2)
9. $i \leftarrow$ index of the $j$-highest value in $\hat{b}$ and mapped to the corresponding position of information bit
10. $R^{(1)}_t \leftarrow \infty \times (2\hat{u}_{old,j} - 1)$
11. $\hat{u}, L, R \leftarrow$ BP decoder$(L, R)$
12. $t \leftarrow t + 1$
13. if generate training data then
14. $I_1, I_2, c \leftarrow$ input data pre-processing$(L, R, H_{\text{CRC}})$
15. $m \leftarrow$ label Undo training data$(R^{(1)}_t, u_t)$
16. Undo data pool $\leftarrow (I_1, I_2, c, m)$
17. end if
18. if $\hat{u}$ passes CRC or $t == T_{\text{max}}$ then
19. return $\hat{u}$
20. end if
21. if $l < d$ then
22. if Use CNN-aided Undo Model then
23. $I_1, I_2, c \leftarrow$ input data pre-processing$(L, R, H_{\text{CRC}})$
24. $\hat{m} \leftarrow$ CNN-aided Undo model$(I_1, I_2, c)$
25. else
26. $\hat{m} \leftarrow 0$
27. end if
28. if $\hat{m} \leq 0.5$ then
29. $l \leftarrow l + 1$
30. $\hat{u}_{old} \leftarrow \hat{u}$
31. Multiple-Bits BF Subtree$(L, R, \hat{u}_{old}, t, l, (u))$
32. $l \leftarrow l - 1$
33. end if
34. end if
35. end for

The input data and architecture design of the CNN-aided Undo model are as similar as the CNN-aided BF model. The only difference is that the output layer dimension is reduced from $K$ to 1. Thus, the loss function is also revised to binary cross-entropy as below:

$$\mathcal{L}(m, \hat{m}) = -m \log \hat{m} - (1 - m) \log (1 - \hat{m}),$$

where $m$ and $\hat{m}$ denote the labeled data and predicted value for the “Undo” action, respectively.

The detailed algorithm flow for the proposed CNN-aided tree-based multiple-bits BF (CNN-Tree-MBF) mechanism is summarized in Algorithm 1 and Algorithm 2. If CNN-aided Undo model is enabled, our mechanism is abbreviated as CNN-Tree-MBF-Undo. The received signal will first go through its first round of BP decoding. After a pre-set number of iterations...
for BP, the CRC will be utilized to check whether the BP decoding is successful. If not, Algorithm 2 will be recursively performed in line 31 of Algorithm 2 until reaching the maximum depth $d$. The for in line 7 of Algorithm 2 means $w_i$ bits with the highest error probability, predicted by the CNN-BF model, will be sequentially attempted in the descending order of error probability. The if in line 21 and line 28 will jointly determine whether to recursively perform Algorithm 2 to enter the next level and start a new subtree. The line 18 shows that this mechanism will be continuously performed until the CRC is passed or the maximum trials $T_{\text{max}}$ in bit flipping is reached. In addition, suppose the generation of training data is required, the message $u$ must be provided for the data labeling.

V. Simulation Results

In this work, we utilize the recurrent neural network-based belief propagation (RNN-BP) algorithm [17] to replace the conventional BP decoding algorithms. The RNN-BP can dramatically reduce the required number of BP iterations $I$ from 40 to 5, which can dramatically decrease the additional decoding latency caused by each flipping attempt and makes the BF mechanism more practical and efficient. Besides, the syndrome loss is also applied for the training of RNN-BP to optimize the scaling parameters from block-level, which can further enhance the decoding performance of BP algorithm [33]. The code length $N$ and information bits $K$ are set to 64 and 32, respectively. Besides, 11-bit CRC is added as information bits and utilized to check the decoding results. The generator polynomial is $x^{11} + x^{10} + x^9 + x^8 + 1$ as reported in [35]. Besides, the training SNR for both CNN models is set to 1 dB and the testing SNR ranges from 0 dB to 3 dB, which can be used to demonstrate that our models can generalize to unseen SNR values. The simulation setup is summarized in Table I.

<table>
<thead>
<tr>
<th>Encoding ($N, K, r$)</th>
<th>Polar code (64, 32, 11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codeword Construction</td>
<td>Bhattacharya bounds algorithm @ SNR = 0 dB</td>
</tr>
<tr>
<td>Decoding Algorithm</td>
<td>RNN-BP [17]</td>
</tr>
<tr>
<td>Number of BP Iteration (I)</td>
<td>5</td>
</tr>
<tr>
<td>Training SNR (dB)</td>
<td>1</td>
</tr>
<tr>
<td>Testing SNR (dB)</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>CRC Generator Polynomial</td>
<td>$x^{11} + x^{10} + x^9 + x^8 + 1$</td>
</tr>
<tr>
<td>Testing Codeword/SNR</td>
<td>240,000</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Training and Testing Environment</td>
<td>DL library of Keras with 17-6700 CPU and NVIDIA GTX 2080 GPU</td>
</tr>
</tbody>
</table>

A. Comparison of Prediction Accuracy Under Different Numbers of Flipping Attempts

The performance of the CNN-Tree-MBF mechanism highly relies on the prediction accuracy of CNN-BF. The higher accuracy means that our CNN-BF model has better error correction capability to dynamically identify the erroneous bits under limited flipping attempts. Thus, it also plays an important role in reducing the number of flipping attempts, namely the decoding latency. In Fig. 7, we firstly compare the prediction accuracy between the proposed CNN-BF model and state-of-the-arts, namely CS-BF [24] and generalized BF (G-BF) [27]. The number of flipping attempts is the number of tries until the decoding result passes CRC. The prediction accuracy is determined by the number of cumulative successful decodings at the number of flipping attempts as a percentage of the total samples. Note that the evaluation of prediction accuracy is based on 1-bit correctable codewords, which means the codewords can be successfully decoded by only 1 correct flipping attempt. The multiple-bits correctable codewords are not included in this experiment. The maximum number of flipping attempts $T_{\text{max}}$ is set to 12 which is as same as $|\text{CS}|$. Besides, the flipping order for CS-BF [24] and G-BF [27] is based on the descending order of the error rate and the ascending order of $|L_{\text{err}}|$, respectively.

In Fig. 7, the performance of the CNN-BF model without the input data of CRC results $c$ is also compared, which is adopted in [29]. As seen in Fig. 7, all methods have better prediction accuracy as the SNR increases. However, CNN-BF predicts the correct BF position at a significantly better accuracy, especially at the earlier number of attempts, as well as having a higher ceiling for improvement. These outstanding improvements are the result of two reasons. First, the well-trained CNN model has a more accurate BF selection. Although G-BF can also dynamically select flipping bits based on $|L_{\text{err}}|$, it is not accurate enough and suffers from noise at low SNR ranges. Second, CNN-BF can flip bits outside of the critical set which achieves better error correction capability over CS-BF. Both of the reasons contribute to the reduction of 5 flipping attempts for CNN-BF compared to CS-BF and G-BF. Besides, after including the CRC results $c$ as input data, the performance improves significantly with an additional reduction of 5 flipping attempts compared to the CNN-BF model without using CRC results, which confirms our idea. Although CRC can only be used for error-detection, it is informative for CNN models to extract features and identify the error-prone bits. Besides, after transforming into an image, the information in the temporal domain can be integrated with
improved prediction accuracy and thus reduce the decoding latency caused by BF.

### B. Comparison of Block Error Rate Under Different Numbers of Flipping Attempts

To further quantify the above results, we realize the contribution of prediction accuracy to the block error rate. In the following experiments, the performance is evaluated on arbitrary codewords, which is not constrained to only 1-bit correctable codewords. Thus, it is a real situation and the benefits of multiple-bits BF over 1-bit BF [29] can be revealed. In this experiment, we compare the BLER under different numbers of flipping attempts with SNR set to 1 dB as shown in Fig. 8.

In addition to the performance of our CNN-Tree-MBF and CNN-Tree-MBF-Undo, some baseline references and related BF-based approaches are also included for the comparison. There are four baseline references, including RNN-BP [17], SCL with list size $L = 8$, and CA-SCL with list size set to 4 and 8 [4], which are the straight lines in Fig. 8. For the related BF-based approaches, CS-BF [24], G-BF [27], 1-bit CNN-BF [29], and bit-strengthening BF (BS-BF) [23], are also compared. In this part, we adopt exhaustive CS-BF, which has better decoding performance at the cost of higher flipping attempts. Thus, the maximum number of flipping attempts is set to 2 $\times$ (CS). Moreover, for BS-BF, the maximum flipping attempts is $\log N - 1$, which is corresponding to the number of stages in the BP’s factor graph and is 5 in our case. For both CNN-Tree-MBF and CNN-Tree-MBF-Undo, we compare three different tree structures as indicated behind the label. For example, 5-2-1 means the tree’s maximum depth $d = 3$ and the number of nodes in each level for $w_1$, $w_2$, and $w_3$ is 5, 2, and 1, respectively.

From Fig. 8, we can observe that CA-SCL with list size set to 8 can achieve the best decoding performance. It is due to the most likely paths are kept to avoid the mistakes happening in the early stages and the decoded results can be selected by the CRC mechanism. However, compared to BP decoding algorithms, CA-SCL suffers from high latency and low throughput due to its sequential processing nature. On the other hand, all BF-based approaches based on the RNN-BP decoding algorithm achieves great improvement, but at the sacrifice of longer decoding latency. It demonstrates that the BF mechanism can provide a compromise for adjustment between decoding performance and latency.

From Fig. 8, compared to the work of 1-bit CNN-BF, our proposed CNN-Tree-MBF can significantly reduce the number of flipping attempts under the same decoding performance. Besides, it can outperform CA-SCL with list size $L = 4$. It is due to our proposed approach can benefit from the better error correction capability of multiple-bits BF and higher prediction accuracy of CNN-BF model. Besides, the tree-based Undo model also plays an important role in early terminating the useless flipping attempts. These reasons jointly contribute to the significant improvement over 1-bit CNN-BF [29].

Besides, for CNN-Tree-MBF, the BLER decreases faster as the tree’s maximum depth increases due to its better error correction capability. However, the flipping error at the early level may cause more useless flipping attempts for the deeper tree structure. Thus, the number of nodes in each level decreases as the depth deepens. To best explore the tree structure, the width in each level can be dynamically adjusted by neural networks or based on the magnitude of the model prediction results, which are left as our future work.

Moreover, we can observe that the performance of CNN-Tree-MBF-Undo is slightly worse than CNN-Tree-MBF, this is due to the inaccurate prediction from CNN-aided Undo model, which degrades the error correction capability. However, it is helpful for further decreasing the flipping attempts, which will be analyzed in the following experiments.

### C. Performance Under Different SNRs With Constant Maximum Number of Flipping Attempts

1) **Block Error Rate:** In Fig. 8, we compare the BLER under different numbers of flipping attempts to evaluate the speed of improvement for different approaches. Thus, the parameter of $T_{\text{max}}$ can be dynamically adjusted for the tradeoff between decoding performance and decoding latency to meet different requirements. Now, we evaluate the BLER under different SNRs with $T_{\text{max}}$ set to 24. For clarity, only one tree structure is compared for both CNN-Tree-MBF and CNN-Tree-MBF-Undo, which is set to 5-2-1. From Fig. 9, we can observe that our proposed CNN-Tree-MBF has about 1.5 dB performance gain compared with RNN-BP, which demonstrates that the BP-based decoding algorithms can be effectively improved via BF mechanisms. Compared to state-of-the-arts CS-BF, G-BF and CNN-BF, our approach also has about 0.6 dB performance gain by taking advantage of deep learning techniques and the well-designed multiple-bits BF strategy. For the BS-BF [23], though the authors demonstrate that it can approach SCL with list size $L = 16$ under code length $N = 2048$, it only has a slight improvement in our experiments due to the limited number of flipping attempts. However, our approach can further outperform CA-SCL with $L = 4$, which has been seen as state-of-the-art for the polar decoder. Later, we will show that as $T_{\text{max}}$ increases, our approach can even outperform CA-SCL with $L = 8$.

2) **Average Flipping Attempts:** To evaluate the impact of additional decoding latency caused by BF, we examine the
average flipping attempts $T_{avg}$ for each approach in Fig. 9. From Fig. 10, the average flipping attempts decreases rapidly as SNR increases. For example, at SNR = 3 dB, the flipping attempts for CNN-BF, CNN-Tree-MBF and CNN-Tree-MBF-Undo are merely 0.28, 0.11 and 0.10, respectively. It shows that the average increase in decoding latency is small enough. However, it still contributes to significant improvement in decoding performance as shown in Fig. 9. The analysis of overall decoding latency, including the inference of CNN model, will be provided in Section V.F. From Fig. 10, the average flipping attempts of CS-BF, G-BF and CNN-Tree-MBF are very close, this can be indicated in Fig. 8. Because the average flipping attempts can be calculated by integrating the BLER over the number of flipping attempts. As the number of flipping attempts increases, the BLER decreases slowly and thus the uncorrectable codewords will dominate the average flipping attempts, which make them have similar performance. Suppose $T_{max}$ is set smaller, the gap between them will become more apparent. On the other hand, compared to CNN-BF, our CNN-Tree-MBF can significantly reduce the average flipping attempts by 41% due to the rapidly decreased BLER and the better error correction capability as shown in Fig. 8. Moreover, our CNN-Tree-MBF-Undo can utilize CNN-aided Undo model for early termination to further reduce 19% average flipping attempts at the cost of slightly degraded BLER. For the BS-BF, it has the lowest average flipping attempts due to $T_{max}$ is only 5.

D. Analysis Between Tree-Based Undo Model and CNN-Aided Undo Model

The proposed tree-based Undo model is a criteria-based model that performs the “Undo” action only when the flipping attempt reaches the maximum depth; otherwise, it successively flips another bit. On the other hand, before reaching the maximum depth, if the CNN-aided Undo model is applied, it can dynamically determine the action by examining the metadata $L$, $R$ and CRC results $c$ from the last BP decoding. Thus, it can effectively avoid the useless flipping attempts but slightly degrade the error correction capability. For a better evaluation of the pros and cons of these two strategies, the confusion matrices of them are shown in Fig. 11, which is greatly helpful for the explanation of our previous experiments.

In this experiment, the performance is evaluated based on the codewords that have not reached the maximum depth. Thus, the last flipped bit is always considered as the correct position and thus the “No Undo” action is always performed by the tree-based Undo model as shown in Fig. 11. The meaning behind TP, TN, FP, and FN can refer to Section IV.B. The higher TP and TN can reduce the number of useless flipping attempts and avoid degrading the error correction capability, respectively. Vice versa, the higher FN and FP will result in useless flipping attempts and degrade the error correction capability, respectively. From Fig. 11, because the tree-based Undo model always performs “No Undo” action, it sacrifices TP to obtain 100% TN. On the other hand, the CNN-aided Undo model strikes a balance between TP and TN, which improves TP to avoid useless flipping attempts at the cost of increased FP. Thus, it’s confusion matrix can clearly illustrate the simulation results shown in Fig. 9 and Fig. 10. Meanwhile, it also demonstrates that there is still room for future extension to further enhance the prediction accuracy of the CNN-aided Undo model.

E. Performance Under Different SNRs With Various Maximum Number of Flipping Attempts

1) Block Error Rate: We repeat the experiments in Fig. 9 and Fig. 10 but at the various maximum number of flipping attempts $T_{max}$ to demonstrate its great flexibility to dynamically adjust
between decoding performance and decoding latency. In Fig. 12, the performance of RNN-BP, SCL, and CA-SCL is also derived as the baseline references. However, for clarity, G-BF, BS-BF and CNN-BF are removed. On the other hand, the performance of CS-BF under different number of flipped bits $\omega$, corresponding to $2^\omega \times |\text{CS}|$ maximum number of flipping attempts, is also derived for a fair comparison. Due to the maximum number of flipping attempts increases, we increase the tree’s maximum depth to 4 to better exploit the benefit of multiple-bits BF with the structure set to 14-2-1-1. From Fig. 12, we can observe that the proposed CNN-Tree-MBF has similar performance as CA-SCL with $L = 4$ and 8, which improves about 1.4 dB and 1.8 dB compared to RNN-BP when $T_{\text{max}} = 12$ and 96, respectively. Meanwhile, when $T_{\text{max}} = 12$, it can outperform CS-BF with $\omega = 3$ which is equivalent to $T_{\text{max}} = 96$. Thus, our approach can significantly improve the error correction capability with less flipping attempts. Besides, it also provides a wide range for the adjustment of the desired BLER by setting the different number of $T_{\text{max}}$.

2) Average Flipping Attempts: Then, the average flipping attempts for different approaches are shown in Fig. 13. We can observe that the additional decoding latency for our proposed CNN-Tree-MBF is still small enough even for $T_{\text{max}} = 96$. However, it can still contribute to significant improvement as shown in Fig. 12. Besides, our proposed CNN-Tree-MBF with $T_{\text{max}} = 12$ can outperform CS-BF with $\omega = 3$ at about 89% reduction in average flipping attempts.

F. Analysis of Computational Complexity, Memory Overhead, and Decoding Latency

Finally, we further analyze the computational complexity, memory overhead, and decoding latency between our proposed CNN-Tree-MBF and the related works, including RNN-BP [17], CS-BF [24], and CA-SCL [4], which are summarized in Table II. To achieve comparable performance as CA-SCL with $L = 8$, we set $T_{\text{max}} = 96$ for analysis. Note that the compared CA-SCL [4] is a general case and many hardware optimization techniques can be found in [36]–[37] to increase throughput and reduce decoding latency.

1) Computational Complexity: We use the number of floating-point operations (FLOPs) to represent computational complexity. The FLOPs for each RNN-BP decoding are $2IN$ and the overall FLOPs for CS-BF increase linearly with $T_{\text{avg}}$. For our proposed CNN-Tree-MBF, it has additional overhead for the inference of CNN model, which increases linearly with the average times of CNN inference $T_{\text{CNN}}$ and thus is based on the tree structure as shown in Fig. 6(a). Compared to other CNN models used in computer vision with multiple GFLOPs, our model is tiny with only 9.8M FLOPs. However, it is still greater than other conventional approaches.

2) Memory Overhead: We use the number of parameters to evaluate memory overhead. RNN-BP stores $\mathbf{L}$, $\mathbf{R}$ messages for the calculation of updated messages, which is about $N(n+1)$ because the memory can be shared between $\mathbf{L}$, $\mathbf{R}$ messages [38]. CS-BF requires additional memory for the storage of the flipping table, which is around $\omega T_{\text{max}}$. Our proposed CNN model requires the storage of both $\mathbf{L}$, $\mathbf{R}$ messages in each iteration as the input data, which is $2^\omega$ times than RNN-BP. Besides, it also requires additional memory for the prediction results, the decoded bit value for the determination of flipping direction, and the storage of CNN parameters, which is $T_{\text{max}}$, $K$ and around 0.3M, respectively. For the case of CA-SCL, we refer to [24], which is $[N+(N-1)L]+L+(2N-1)L$.

3) Decoding Latency: We evaluate the decoding latency in terms of the required number of time steps and CPU computing time. For the iterative RNN-BP decoding process, it requires $2In$ time steps and thus CS-BF is $2InT_{\text{avg}}$. Our CNN model requires additional time steps for the inference which is dependent on the number of NN layers [39]–[40] and is 10 in our case. Besides, it also increases linearly with $T_{\text{CNN}}$. For a general CA-SCL, it requires $2N-2$ time steps. The CPU computing time is also listed in Table II. Note that the sorting time of the output probabilities is omitted because it is far smaller than the model inference and BP decoding time. From Table II, it shows that our approach can achieve comparable performance as CA-SCL with fewer time steps and shorter computing time at high SNR.

In summary, although our CNN model consumes more hardware resources, many techniques have been proposed, such as pruning and quantization, which can successfully compress the neural networks by more than 40 times without performance
TABLE II
ANALYSIS OF COMPUTATIONAL COMPLEXITY, MEMORY OVERHEAD, AND DECODING LATENCY

<table>
<thead>
<tr>
<th>SNR</th>
<th>Floating-Point Operations (FLOPs)</th>
<th>Memory Overhead</th>
<th>Decoding Latency (Time Steps)</th>
<th>CPU Computing Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6dB</td>
<td>2IN + 3840</td>
<td></td>
<td>2IN + 60</td>
<td>0.0015</td>
</tr>
<tr>
<td>3dB</td>
<td>2IN + 736</td>
<td></td>
<td>2IN + 1047 + 0.00157</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For N = 64, K = 32, n = 6, I = 5, L = 8, ω = 3, T_{max} = 96

Fig. 14. Comparison of prediction accuracy between the proposed CNN-BF and the opposite CS-BF [24] under different numbers of flipping attempts.

Fig. 15. Comparison of block error rate between different approaches under different numbers of flipping attempts with SNR = 1 dB.

degradation. These techniques can be applied to our CNN model for the reduction of hardware complexity. For the future extension, we suggest that the CNN model can be replaced by some lightweight ML models, such as tree-based models, for the tradeoff between complexity and prediction accuracy.

G. Evaluation of Scalability

To demonstrate the scalability of our proposed approach, we also repeat the simulations by increasing code length from 64 to 128 in Fig. 14 to Fig. 16. We can observe that the simulation results under N = 128 have a similar trend as N = 64. Besides, the gap between our proposed CNN-Tree-MBF and CS-BF becomes larger, which demonstrates the great advantage of neural networks under more difficult conditions.

VI. CONCLUSION

In this paper, we present a novel convolutional neural network-aided tree-based bit-flipping decoder, which can effectively exploit the benefit of multiple-bits BF. With carefully designed input data and domain-specific data pre-processing, our model can learn from the BP metadata and the CRC results to correctly predict flipping position, with more accuracy than the prior critical set method and our previous CNN-BF. Besides, the tree structure for flipping strategy can strike a good balance between the benefit of multiple-bits BF and the increased flipping attempts caused by wrongly flipped bits. Therefore, it can achieve more than 1.5 dB gain over RNN-BP algorithm with slightly increased decoding latency.

REFERENCES

Chieh-Fang Teng (Student Member, IEEE) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2017. He is currently working toward the Ph.D. degree with the Graduate Institute of Electronics Engineering, National Taiwan University. His research interests include the areas of Internet of Things, VLSI architecture for DSP, and machine learning assisted wireless communication systems design.

An-Yeu (Andy) Wu (Fellow, IEEE) received the B.S. degree from National Taiwan University in 1987, and the M.S. and Ph.D. degrees from the University of Maryland, College Park in 1992 and 1995, respectively, all in electrical engineering. In August 2000, he joined the faculty of the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan, in 2017. He is currently a Distinguished Professor. His research interests include VLSI architectures for signal processing and communications, and adaptive/multirate signal processing. He has authored or coauthored more than 190 refereed journal and conference papers in above research areas, together with five book chapters and 16 granted US patents. From August 2007 to December 2009, he was on leave from NTU and served as the Deputy General Director of SoC Technology Center (TSC), Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan. In 2010, he was the recipient of the Outstanding EE Professor Award from The Chinese Institute of Electrical Engineering (CIEEE), Taiwan. From 2012 to 2014, he served as the Chair of VLSI Systems and Applications (VSA) Technical Committee (TC), one of the largest TCS in IEEE Circuits and Systems (CAS) Society. In 2015, he was elevated to IEEE Fellow for his contributions to DSP, VLSI, and SoC designs for communication IC/SOC. From 2016 to 2019, he served as Director of Graduate Institute of Electronics Engineering (GIEEE), National Taiwan University. He is currently the Editor-in-Chief (EiC) of IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS (JETCAS).