TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering

Improving the Decoding of Error Correction Code via Machine Learning

by

Yuan Y.

MSc Thesis

Thesis committee

Chair:	Prof., A. Alvarado
Member 1:	Assistant Prof., I. Nikoloska,
Member 2:	Assistant Prof., A.K. Balatsoukas Stimming
Advisory Member 3:	Dr. Y.C.G. Gultekin

Graduation

Program:	Artificial Intelligence & Engineering Systems
Research group:	SPS-ICT Lab
Thesis supervisor:	Prof. A. Alvarado
Date of defense:	02-09-2024
Student ID:	1778307
Study load (ECTS):	120
Tracks:	High-tech systems and robotics

Contents

Improving the Decoding of Error Correction Codes via Machine Learning

Yuncheng Yuan
Eindhoven University of Technology
y.yuan@student.tue.nl

***Abstract*—This report explores the application of neural networks (NNs) as effective decoders for error correction codes (ECCs). First, our study replicates two types of NN decoders from the literature: (i) single-label neural network (SLNN) decoder and (ii) multi-label neural network (MLNN) decoder. Upon pruning the NN architecture, it was determined that the SLNN decoding is equivalent to maximum likelihood decoding. Second, this report introduces a novel MLNN structure characterized by its minimal complexity. Third, we reproduced two transformer-based decoders, the error correction code transformer and the cross-attention message passing transformer. We observed that both transformer-based decoders perform worse than information set decoding.**

***Index Terms*—Error correction codes, machine learning, deep learning, large language model, neural network decoder, maximum likelihood.**

I. Introduction

Error correction codes (ECCs) are an integral part of telecommunication systems, ensuring reliable data transmission through noisy channels. The first ECC, the Hamming $(7,4)$ code, was introduced by Hamming in 1950 [1]. The development of telecommunication systems has since led to the development of more sophisticated codes that can better protect longer data sequences, e.g., Bose–Chaudhuri–Hocquenghem (BCH) codes [2], Golay codes [3], polar codes [4], etc. These codes are widely implemented across various applications.

The ultimate objective of a decoder for an ECC is to achieve soft-decision maximum likelihood (SDML) decoding performance while minimizing computational complexity and latency [5]. Although SDML decoding provides optimal performance, it is computationally intensive and introduces significant latency, especially for longer codes. For instance, for the relatively short BCH $(31,21)$ code, which has 21 information bits and 31 encoded bits, the exhaustive generation of 2^{21} possible codewords and the calculation of Euclidean distance between these codewords and the received sequence required to realize SDML decoding is already infeasible. To overcome SDML limitations, various suboptimal decoding algorithms have been proposed for different classes of codes. The belief propagation (BP) algorithm, used for low-density parity-check (LDPC) codes [6], for instance, creates a trade-off between the decoding performance and computational complexity, offering near-optimal error correction without the complexity of exhaustive search methods.

Recently, decoders based on neural networks (NNs) and deep learning (DL) have emerged as viable alternatives to traditional decoding algorithms, providing performance similar to SDML decoders without requiring exhaustive search [7]. The BP-based NN decoder retain BP's characteristic but results in more accurate estimates compared to general NN decoders. [8]. However, the efficiency of NN-based decoders depends on the complexity of the NN architecture. Increased number of hidden layers and neurons per layer, although improve the decoding performance, lead to higher computational demands, decoding latency, and reduced energy efficiency.

To address this problem, researchers have studied on pruning techniques in NN training [9]. Pruning reduces the network size by reducing model size before or after training. Strategies such as magnitude-based pruning discard weights with the smallest absolute values [10]. Pruning has also been applied to large language models (LLMs) to expedite training and inference [11], manage memory constraints on mobile devices, and accelerate training for expanding networks [12].

LLMs have recently become prevalent, particularly in natural language processing (NLP) and computer vision (CV). These models, leveraging advanced architectures like the transformer model introduced in [13], handle complex tasks involving human language understanding and visual data interpretation. The transformer's ability to manage long-range dependencies (i.e., time correlations) and uncover intricate data relationships is beneficial for tasks with noisy or incomplete information, as the transformer can discern and learn underlying patterns despite noise.

Recently, significant advancements in decoding ECCs have been achieved with transformer-based architectures. The error correction code transformer (ECCT) enhances decoding capabilities using the self-attention mechanism [14]. The cross-attention message passing transformer (CrossMPT) employs a cross-attention mechanism for improved decoding performance [15]. Both act as soft-input decoders, prioritizing and flipping likely erroneous bits to improve accuracy. These models outperform traditional algorithms like BP but require a parity check matrix for self-attention masking, making them non-model-free. However, their high computational complexity and extensive training times make them unsuitable for low-power and low-latency applications.

This paper further investigates and optimizes single label neural network (SLNN) and multi-label neural network (MLNN) decoders to reduce their computational complexity

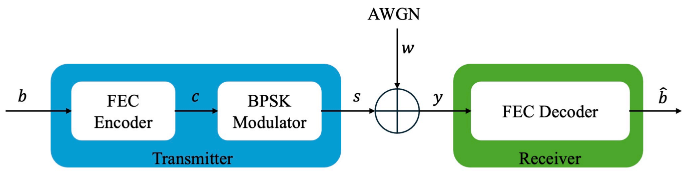


Fig. 1. Overview of the communication system containing a transmitter, AWGN channel, and a receiver.

and improve their performance. Then, ECCT and CrossMPT results are reproduced and analyzed. The main contributions are:

1) We pruned the SLNN structure and proved that the optimal SLNN decoder does not require training to achieve SDML results.
2) We proposed a new low-complexity MLNN architecture based on the SLNN decoder, demonstrating lower complexity and better performance than MLNN in all encoding method.
3) We reproduced ECCT and CrossMPT results and compared to traditional decoding algorithms, identifying limitations of current transformer-based decoders.

The remainder of this paper is organized as follows. Section II introduces the necessary preliminary theory and notations, and presents the system model under consideration. SLNN and MLNN decoders are investigated in Sec. III and Sec. IV, respectively, including their theoretical derivations, simulation results, and complexity comparisons with previous works. Section V and Sec. VI analyze the performance and limitations of ECCT and CrossMPT, respectively. Finally, Sec. VII concludes the paper and outlines future prospects.

II. Methodology

A. System Model

We consider a simplified communication system model to investigate NN decoders. As illustrated in Fig. 1, this model comprises a transmitter, the additive white Gaussian noise (AWGN) channel, and a receiver. The transmitter employs a ECC Encoder coupled with a binary phase-shift keying (BPSK) modulator. Information bits, denoted as $\boldsymbol{b} = (b_1, b_2, \ldots, b_k)$, are encoded into codewords $c = b\mathbf{G}$, where $\mathbf{G}$ is a $n \times k$ generator matrix related to linear block code. Given the encoding of k information bits, the system can produce a total of 2^k distinct codewords.

BPSK modulation maps binary codewords c_i to symbol sequences using $s_i = 2c_i - 1$. These symbols are transmitted through an AWGN channel, where normally distributed noise $\mathcal{N}(0, \sigma^2)$ is added, with the noise variance $\sigma^2 = \frac{N_0}{2}$. Here, N_0 represents the noise spectral density, which is used to calculate the noise variance. In this context, the signal-to-noise ratio (SNR) is often expressed in terms of the noise spectral density N_0 and the energy per bit E_b, commonly denoted as

$$\text{SNR} = \frac{E_b}{N_0} \tag{1}$$

This formulation is particularly useful in digital communication systems, where E_b/N_0 serves as a normalized measure of signal strength relative to noise, facilitating performance comparisons across different modulation schemes and coding techniques.

At the receiver, the transmitted codeword $\hat{b}$ is estimated by decoding the received noisy symbols, where $y = s + w$. The SDML decoder, an exhaustive search algorithm, identifies the codeword in the codebook that maximizes the likelihood or minimizes the Euclidean distance relative to the received signal. This process involves evaluating the likelihood or distance for all possible codewords, making the SDML decoder computationally intensive. While the SDML decoder achieves optimal performance and serves as a benchmark, its high complexity and resource demands have prompted researchers to explore alternative solutions that offer reduced latency and complexity, such as BP decoders, NN decoders, and transformer-based decoders.

B. Soft-decision Maximum Likelihood Decoding

For the AWGN channel, the ML decoding can be expressed in terms of minimizing the Euclidean distance or equivalently, maximizing the correlation (dot product) between $\mathbf{y}$ and $\mathbf{s}$, described as

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} P(\mathbf{y}|\mathbf{s}), \tag{2}$$

The SDML decoder, used as the benchmark in this study, operates directly on the real-valued received symbols $\mathbf{y}$, as soft-input decoder. The SDML decoder selects the estimated transmitted codeword $\hat{\mathbf{c}}$ from the codebook $C = \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n\}$, where n represents the block length of possible codewords. Thus, the decoder identifies the most likely transmitted codeword $\hat{\mathbf{c}}$ by selecting the codeword that exhibits the highest correlation with the received vector. Consequently, the SDML decoder functions by evaluating the correlation between each codeword in the codebook and the transmitted codeword. By selecting the codeword that maximizes correlation or minimize Euclidean distance, the decoder identifies the most probable transmitted codeword $\hat{\mathbf{c}}$.

C. Performance Metrics

The decoder makes an error when a codeword $\mathbf{c}$ is transmitted, but the decoder outputs $\hat{\mathbf{c}} \neq \mathbf{c}$. The block error rate (BLER) is defined as

$$\text{BLER} := \frac{1}{D}\sum_{j=1}^{D} \mathbb{1}(\hat{c} \neq c^{(j)}), \tag{3}$$

where D represents the number of message samples. The $c^{(j)}$ denotes the j-th encoded message in the form of a codeword. The index j ranging from 1 to D, uniquely indexing each encoded message within the set of D samples. When $\hat{c} \neq c^{(j)}$, indicator function equals 1.

Similarly, the bit error rate (BER) is defined as

$$\text{BER} := \lim_{D\to\infty} \frac{1}{kD}\sum_{j=1}^{D}\sum_{i=1}^{k} \mathbb{1}(\hat{b}^{(j)}(i) \neq b^{(j)}(i)), \tag{4}$$

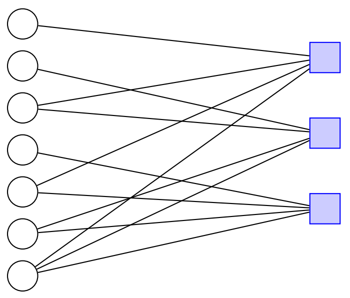


Fig. 2. Tanner graph of the Hamming (7,4) code: circles are check nodes and squares are variable nodes.

BLER and BER are essential metrics for assessing the accuracy of decoders.

D. Belief Propagation

BP is an iterative algorithm used to decode messages in ECCs [16]. The algorithm operates on a Tanner graph, illustrated in Fig 2, which is derived from the parity-check matrix of the code. In this graph, variable nodes correspond to the bits of the codeword, while check nodes represent the parity-check equations. BP involves passing messages between variable nodes and check nodes iteratively. These messages are updated based on the incoming information from neighboring nodes, allowing the algorithm to progressively refine its estimates of the transmitted message. The process continues until convergence is achieved or a predefined number of iterations is reached, resulting in a decoded message that ideally matches the original transmitted data.

The performance of the BP decoder for the Hamming $(7, 4)$ code is shown in Fig 3. The results demonstrate that the BP decoder surpasses the hard-decision maximum likelihood (HDML) decoder, where the input bits are converted to either 0 or 1 before applying the maximum likelihood decoding process. This approach does not consider the probability or confidence of the received values. Although the BP decoder is more effective than the HDML, it falls short of achieving the performance level of the SDML decoder, which utilizes probabilistic information for decoding.

E. Neural Networks

NN is a deep learning structure to mimic human brain. NNs have been successfully applied to several fields including NLP and CV [17]. NNs are composed of interconnected neurons, where each neuron receives multiple inputs, multiplies them by a weight matrix, sums the results with a bias, and then passes the outcome through an activation function to generate an output. The fundamental equation governing a single neuron can be expressed as in Fig 4

$$y_{\text{NN}} = f\left(\sum_{i=1}^{m} w_i^{\text{NN}} x_i^{\text{NN}} + b_{\text{NN}}\right), \tag{5}$$

where y_{NN} is the output of the neural network, f is the activation function, x_i^{NN} represents the inputs, w_i^{NN} are the

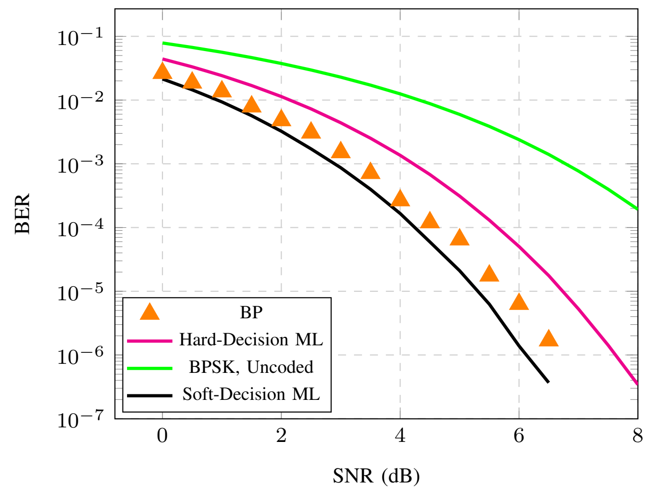
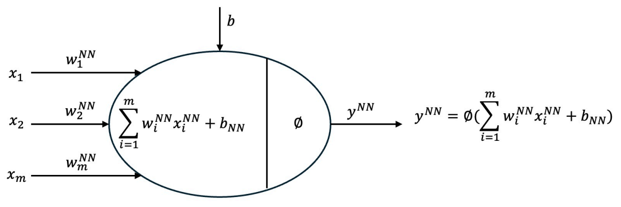


Fig. 3. SNR vs. BER performance of BP for the Hamming $(7, 4)$ code.



b
x_1
x_2
x_m
w_1^{NN}
w_2^{NN}
w_m^{NN}
$\sum_{i=1}^{m} w_i^{NN} x_i^{NN} + b_{NN}$
$\emptyset$
y^{NN}
$y^{NN} = \emptyset(\sum_{i=1}^{m} w_i^{NN} x_i^{NN} + b_{NN})$

Fig. 4. Neuron equation

corresponding weights, b_{NN} is the bias term, and m is the number of inputs.

Each layer in a neural network consists of a set of neurons, where each neuron performs a weighted sum of its inputs, adds a bias term, and then applies an activation function to produce its output. This output is then propagated forward as input to the neurons in the subsequent layer. shown in Fig 5.

During training, a loss function measures the difference between the network's predicted output and the actual target to update parameters. Two commonly used loss functions for binary or multi-class classification problems are the cross-entropy (CE) loss and the binary CE (BCE) loss. The BCE loss used for binary classification tasks is defined as

$$\mathcal{L}_{\text{BCE}}(\mathbf{y}, \mathbf{p}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right), \tag{6}$$

where N is the number of samples, y_i is the actual label, and p_i is the predicted probability for the i-th sample. The CE loss used for multi-class classification tasks is given by

$$\mathcal{L}_{\text{CE}}(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \log(p_{i,c}), \tag{7}$$

where C is the number of classes, $y_{i,c}$ is the c-th element of the one-hot coded vector $\mathbf{y}_i$ indicating if class label c is the correct classification for sample i, and $p_{i,c}$ is the predicted probability that sample i is of class c.

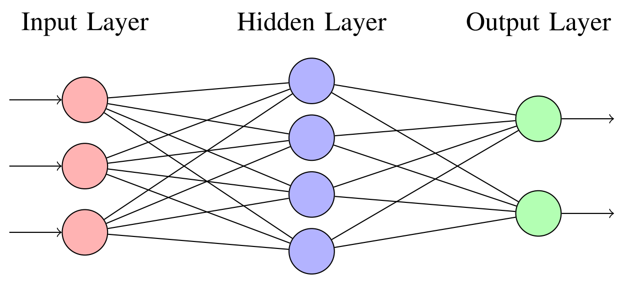


Fig. 5. Example of a fully connected NN with three input neurons, five hidden neurons, and two output neurons.

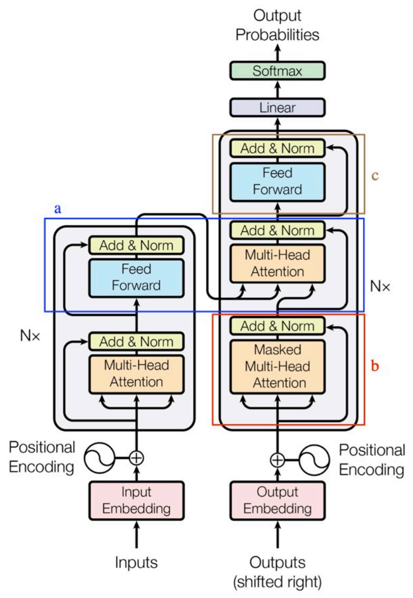


Fig. 6. The transformer architecture. (a) cross-attention mechanism. (b) multihead self-attention mechanism (c) FFNN

F. Post-pruning

Post-pruning is a technique employed to reduce the complexity of neural networks by simplifying the architecture post-training. This process entails the removal of redundant neurons or layers, effectively decreasing the model's complexity while maintaining performance. In NN model, iterative pruning and retraining are commonly utilized. This method systematically prunes a portion of the smallest weights after each training iteration, followed by retraining to fine-tune the remaining weights. The procedure is repeated iteratively until the network achieves the desired level of sparsity.

G. Transformer

Transformer is an attention-based models for machine translation, which are widely used in NLP [18], and CV [19]. Transformer-based architectures can be explained using five important concepts as shown in Fig. 6: (1) initial embedding, (2) multihead self-attention mechanism, (3) cross-attention mechanism, (4) feedforward NN (FFNN), and (5) loss function.

1) Initial Embedding: Input embedding is an essential technique in transformers, transforming discrete input data into continuous, high-dimensional vectors. This embedding process maps each input item to a dense vector in a high-dimensional space, effectively capturing the relationships within the data.

H. Multihead Self-Attention Mechanism

The self-attention mechanism is a key component of the transformer architecture, enabling the model to assess the significance of different parts of the input data during sequence encoding. Unlike RNN, self-attention operates solely on the input sequence. To capture diverse aspects of the input, the transformer employs a multihead attention mechanism, which divides the input into multiple heads. Each head independently performs the attention operation, and the resulting outputs are concatenated and linearly transformed to produce the final output. The multihead attention mechanism for each head i is mathematically expressed as:

$$\text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i, \quad (8)$$

where X denotes the input sequence, and the query ($Q_i = XW_i^Q$), key ($K_i = XW_i^K$), and value ($V_i = XW_i^V$) vectors are derived using learned weight matrices W_i^Q, W_i^K, and W_i^V specific to the i-th head. Here, d_k represents the dimension of the key vectors. The query vector determines the input segments to prioritize, the key vector encapsulates the input itself, and the value vector contains the relevant information used in the attention computation.

After computing the attention scores, the outputs from all heads are concatenated and linearly transformed to produce the final output of the multihead attention mechanism:

$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \quad (9)$$

where W^O is the output projection matrix. This multihead attention mechanism enables the transformer to jointly attend to information from different representation subspaces at various positions within the input sequence, thus enhancing the model's ability to capture complex dependencies and relationships in the data.

1) Cross-attention Mechanism: Self-attention and cross-attention differ primarily in their scope of operation. Self-attention operates within a single sequence, allowing each element to attend to others within the same sequence to understand dependencies and contextual relationships. Cross-attention involves two different sequences, enabling elements of one sequence to attend to elements of another. This distinction is crucial for applications like translation, where the model needs to align and integrate information from distinct sequences to ensure accurate translation.

2) Feedforward Neural Network: In Fig. 6, the FFNN transforms the input data by providing non-linear transformations, enhancing the model's capacity to capture complex patterns. The output of the FFNN is typically followed by a residual connection and layer normalization, ensuring stable training and facilitating better gradient flow through the network as

$$\text{FFN}(x) = x + \text{Linear}_2(\text{ReLU}(\text{Linear}_1(x))), \quad (10)$$

where, Linear_1 and Linear_2 are the two linear networks, and rectified linear unit (ReLU) is the activation.

3) Loss Function: In transformers, the most commonly used loss function for supervised learning is the CE function. The objective is to update the trainable parameters, including $\mathbf{W}$ and $\mathbf{b}$, to minimize the loss

$$\mathcal{L} = -\sum_{i=1}^{n} \tilde{z}_i \log(f_\theta(y)) + (1 - \tilde{z}_i) \log(1 - f_\theta(y)), \quad (11)$$

where $f_\theta()$ represents the model.

III. Single Label Neural Network Decoder

In this section, the SLNN decoder is investigated. First, the concept of the SLNN decoder is introduced. Then, the SLNN performance is presented for two different FEC codes. At last, a pruned SLNN architecture is proposed.

A. NN-based Decoders

Recently, NNs have been applied in the field of decoding ECCs. There are two types of NN decoders. The first type is a model-free NN decoder, suitable for various encoding methods [20]. The second type is an NN decoder designed based on the parity-check matrix of the code and, also known as the neural BP (NBP) decoder [21]. In the model-free NN decoder, the number of neurons in the hidden layer depends on the network structure. As the codeword length increases, the output layer or hidden layer in the model-free NN decoder becomes more complex, requiring more neurons, which complicates the model structure. The NBP decoder addresses these issues by making the hidden layers less complex and more explainable [22].

B. SLNN Architecture and Training

The SLNN decoder was initially described in [23]. The number of neurons in the input layer is fixed to the block length of the noised message y. The design parameters of the NN include the number of hidden layers I and the number of neurons in each hidden layer N_I^h. For the output layer, the number of neurons in output layer is equal to number of codeword 2^k in the codebook. For the hidden layer, the activation function employed is the rectified linear unit (ReLU), defined as $\text{ReLU}(x) = \max(0, x)$. For the output layer, the activation function employed is the Softmax $\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$. The details of the activation functions and the node configuration are given in Table I.

During the training, the SLNN model was trained with $SNR_{\text{train}} = 0$ dB which is inherently low-latency and highly efficient. A low-latency decoder, by definition, requires minimal or no training time, as seen with traditional decoders like the BP decoder, which operates without the need for extensive training. At this SNR level, where signal power is equivalent to noise power, the model is subjected to a challenging environment, forcing it to learn robust decoding strategies effective across a wide range of SNRs. This training condition enhances the model's ability to generalize in diverse noise environments, ensuring that it remains resilient even under varying conditions.

TABLE I
The activation functions and number of neurons selected in the SLNN decoder

Layer	Activation	Number of Neurons
Input Layer (Layer 0)	-	$N_0 = n$
Hidden Layer (Layer $l = 1, \ldots,$ L)	ReLu	N_l^h
Output Layer (Layer $l + 1$)	Softmax	$N_{L+1} = 2^k$

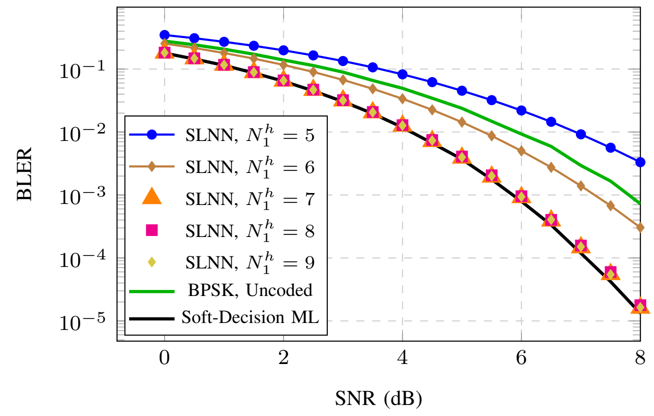


Fig. 7. Hamming $(7,4)$: SNR vs. BLER performance of SLNN in different number of nodes with 1 hidden layer.

Furthermore, by training in such a difficult scenario, the network reduces its dependency on high SNR conditions and can maintain low-latency operation by efficiently processing signals with minimal computational overhead. This efficiency is crucial in real-time applications, where quick, accurate decoding is necessary.

C. Results and Discussion

1) Hamming $(7,4)$ *Code:* As illustrated in Fig. 7, the BLER of the SLNN decoder matches the performance of SDML with a configuration of $l = 1$ hidden layer comprising $N_1^h = 7$ nodes for the Hamming $(7,4)$ code. Furthermore, it was observed that the number of neurons in the hidden layer must be $N_l^h \geq 7 = n$ to achieve a performance comparable to that of the SDML.

2) Parity $(26,10)$ *Code:* Authors of [23] introduced a new FEC code designated as Parity $(26,10)$, which comprises 10 information bits and 16 parity check bits. A critical examination of their generator matrix, detailed in Table II, reveals that generator matrix is not full rank. In Fig. 8, the SLNN is evaluated for Parity $(26,10)$ code in terms of BLER, demonstrating that the network achieves SDML performance when configured with one hidden layer and $N_1^h = 24$. The presence of two hidden layers, provides no improvement in SLNN performance, adding unnecessary complexity to the SLNN decoder.

TABLE II
PARITY (26,10) GENERATOR MATRIX

1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	1	1	0	1
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0	1	1	0	1	1	1
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	1	0	0	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	0	1	1	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1

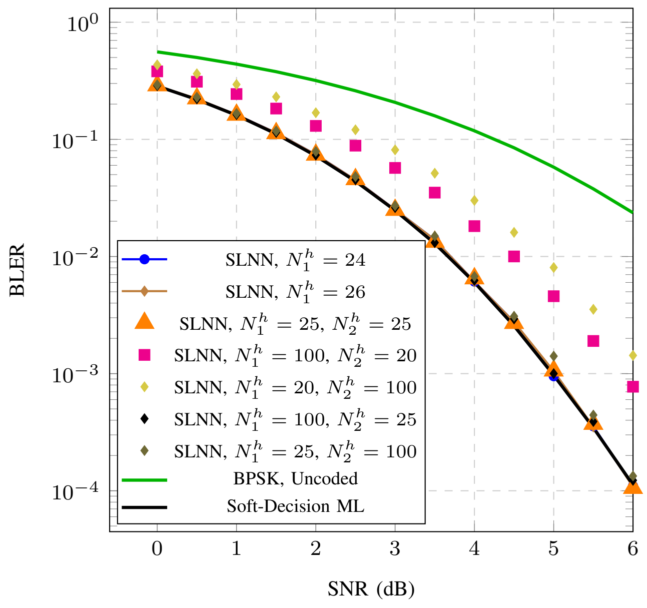


Fig. 8. Parity $(26, 10)$: SNR vs. BLER to compare the performance of the SLNN in different nodes and layers.

D. Pruning the SLNN Architecture

The fully connected SLNN exhibits high computational complexity. However, post-pruning, a significant reduction in the number of edges is observed without performance sacrifice, illustrated in Fig. 9. Specifically, in the Hamming $(7, 4)$ SLNN, only 7 edges in the hidden layer are retained. Figure 10 demonstrates that each input layer node is connected exclusively to a single node in the hidden layer, only transferring values to the output layer. In the case of the Parity $(26, 10)$ SLNN, where $N = 26$, only 24 edges are maintained in the hidden layer, as depicted in Fig. 11. The absence of connections from the bits indicates that these do not incorporate any information bits, leading to the removal of their edges without affecting performance.

Overall, the structure of the SLNN is influenced by the Forward Error Correction (FEC) encoding codes. As the number of FEC encoding code increases, the output layer's neurons neurons grows exponentially with k. The pruned SLNN structure demonstrates that the hidden layer merely transfers values to the output layer, rendering it redundant and contributing to increased computational complexity. Consequently, the hidden layer can be removed, leading to a simplified neural network structure, as illustrated in Fig. 13 for the Hamming $(7, 4)$. This simplification has no impact on performance, as shown in Fig. 12.

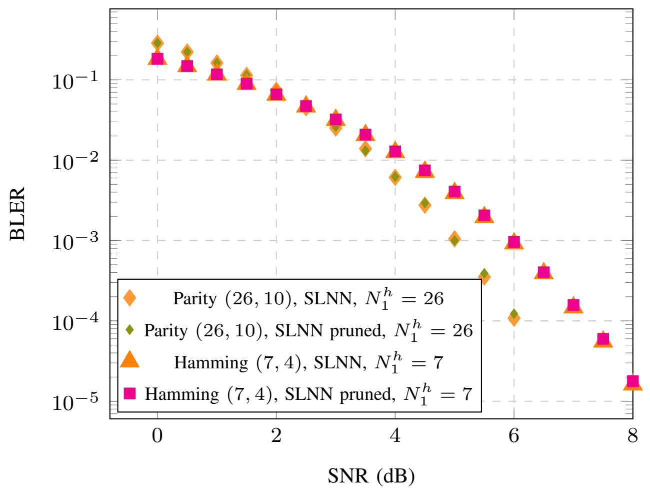


Fig. 9. SLNN: SNR vs. BLER to compare the performance of the SLNN in fully-connected and pruned for Hamming $(7, 4)$ and Parity $(26, 10)$.

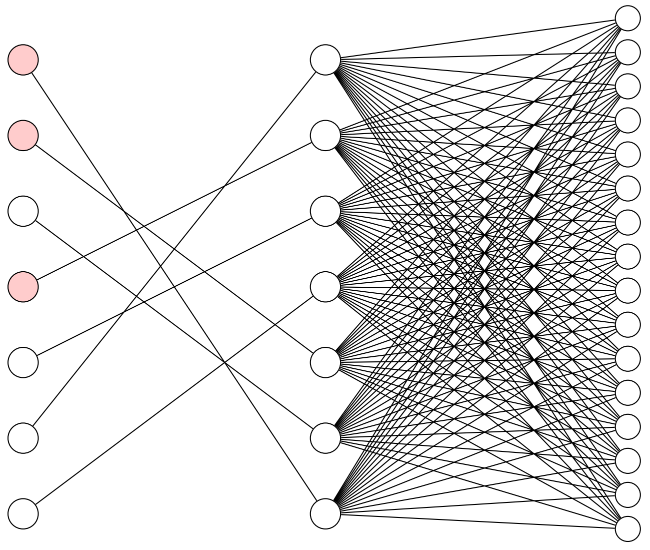


Fig. 10. Hamming $(7, 4)$: SLNN pruned hidden layer: Left side is input layer, white neurons are information bits and red dots are parity check bits.

IV. MULTI LABEL NEURAL NETWORK DECODER

As discussed in Sec. III, the number of output layer neurons in SLNN increases exponentially with k, making it impractical to train large NN decoders, such as for BCH $(63, 51)$, where the network would require 2^{15} trainable parameters. To address this issue, the author of SLNN proposed the MLNN decoder, which reduces the number of output layer neurons by increasing the number of neurons and layers in the hidden layers. This structure allows the real-valued noisy message $\mathbf{y}$ to be directly decoded into k information bits.

A. MLNN Architecture and Training

As described in [23], the MLNN shares similarities with the SLNN, where the number of input layer's neuron corresponds

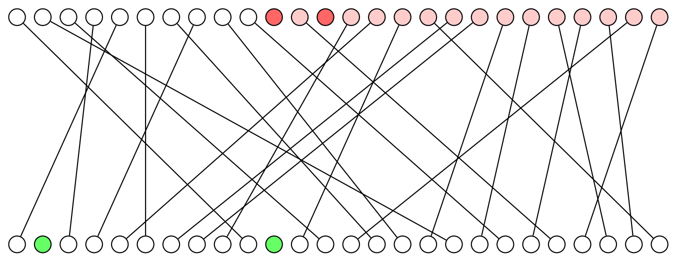


Fig. 11. Parity $(26, 10)$: SLNN pruned hidden layer: on the top is input layer, white neurons are information bits and red dots are parity check neurons, where dark red neurons do not contain any information. Blow neurons are hidden layer and green neurons are isolated.

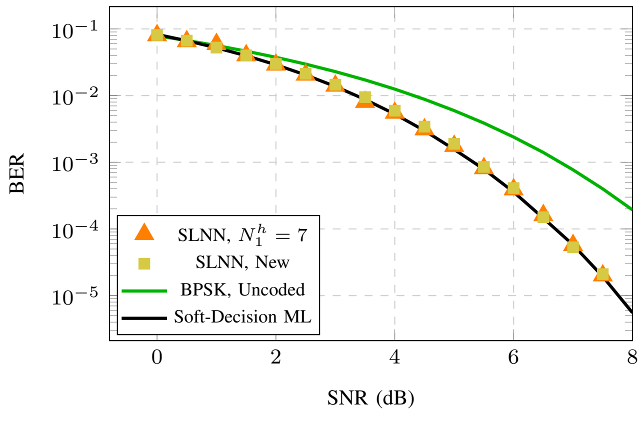


Fig. 12. Hamming $(7, 4)$: SNR vs. BER to compare the performance of the SLNN and new SLNN without hidden layer

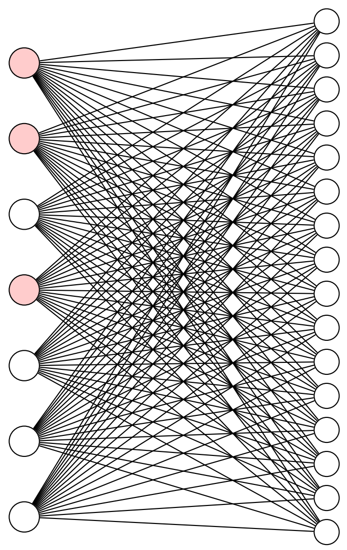


Fig. 13. Hamming $(7, 4)$: SLNN without hidden layer: Left side is input layer, white neurons are information bits and red dots are parity check bits.

TABLE III
THE ACTIVATION FUNCTIONS AND NUMBER OF NEURONS SELECTED IN THE MLNN DECODER

Layer	Activation	Number of Neurons
Input Layer (Layer 0)	-	$N_0 = n$
Hidden Layer (Layer $l = 1, \ldots,$ L)	ReLu	N_l^h
Output Layer (Layer $l + 1$)	Sigmoid	$N_{L+1} = k$

to the block length of the noisy message y. The design parameters of the NN decoder include the number of hidden layers I and the number of neurons in each hidden layer N_I^h. There are no specific limitations on I and N_I^h in the hidden layers; increasing either parameter can enhance performance, albeit at the cost of increased complexity. The output layer contains a number of neurons equal to the number of information bits k. The ReLu activation function is employed in the hidden layers, while the Sigmoid activation function, given by $f(x) = \frac{1}{1+e^{-x}}$, is used in the output layer. Further details on the activation functions and node configurations are provided in Table III.

Same as SLNN, the MLNN was also trained with $SNR_{\text{train}} = 0$ dB and tested for all SNR.

B. Results and Discussion

Compared with SLNN, the MLNN decoder performance closely approach to SDML decoder when configured with two hidden layers and 100 neurons in each layer, as illustrated in Fig. 14. However, for the Hamming $(7, 4)$ code, the MLNN exhibits significantly higher computational complexity than the SLNN. Specifically, the MLNN with $N_1^h = 100$ and $N_2^h = 100$ comprises $11,304$ trainable parameters, whereas the SLNN with $N_1^h = 7$ contains only 184 trainable parameters, necessitating considerably more resources for training.

C. Optimized Structure for the MLNN Decoder

Fig. 10 and Fig. 11 illustrate that the hidden layer of SLNN is redundant. In contrast, the MLNN's primary disadvantage lies in its complex hidden layer. However, the MLNN's output layer, with k neurons, is better suited for longer codewords. To address these challenges, an optimized MLNN architecture with reduced computational complexity is proposed. This new structure integrates the pruned SLNN results with the MLNN architecture. The SLNN's output layer is redefined as the hidden layer in the new MLNN, consisting of 2^k neurons with a Softmax activation function. The output layer retains k neurons and employs a Sigmoid activation function, as detailed in Table IV.

After training with $SNR_{\text{train}} = 0$ dB, the optimized MLNN reaches the SDML performance as shown in Fig. 14. The optimized MLNN with 16 neurons in hidden layer performed better than all regular MLNN models. It also has reduced output layer size compared to SLNN. However, the neuron number of the hidden layer N_I^h will increase exponentially as block length increases, which will cause a high computational complexity.

V. ERROR CORRECTION CODE TRANSFORMER

Although the above newly-proposed MLNN only needs k output nodes, the hidden layer size 2^k grows exponentially in k, which is infeasible to realize for longer codes. To address this, a transformer-based decoder has been proposed

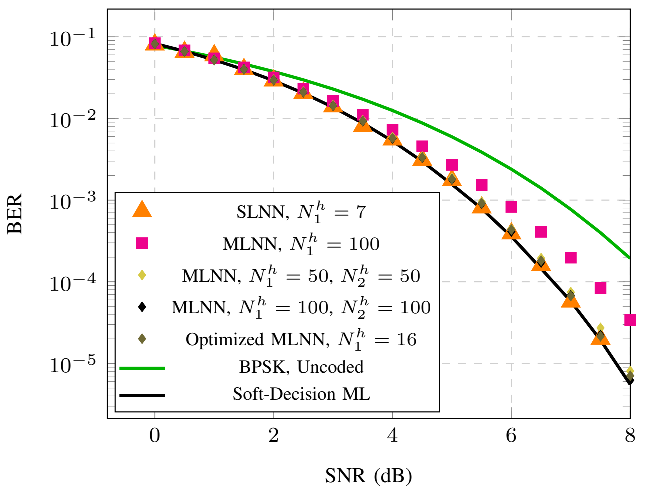


Fig. 14. Hamming $(7, 4)$: SNR vs. BER to compare the performance of the MLNN in different number of neurons and hidden layer. MLNN, $N = 16$ is optimized structure MLNN.

TABLE IV
THE ACTIVATION FUNCTIONS AND NUMBER OF NEURONS SELECTED IN THE OPTIMIZED MLNN DECODER

Layer	Activation	Number of Neurons
Input Layer (Layer 0)	-	$N_0 = n$
Hidden Layer (Layer $l = 1$)	Softmax	$N_{L1} = 2^k$
Output Layer (Layer $l = 2$)	Sigmoid	$N_{L2} = k$

for designing an effective error correction decoder. The self-attention mechanism inherent in transformers is leveraged to identify relationships between inputs and correcting the wrong bits. At the same time, the number of trainable parameters will not exponentially increase as code length increasing. Therefore, ECCT was proposed in [14] outperforms current BP neural network decoders.

A. Architecture and Training

1) Pre-processing: In the pre-processing step, the noisy message y is replaced by a vector of dimensionality $2n - k$, defined as:

$$\tilde{y} = h(y) = [|y|, s(y)]\,, \tag{12}$$

where $[\,,\,]$ denotes vector concatenation, $|y|$ represents the absolute value (magnitude) of y, and $s(y)$ denotes the binary syndrome obtained by multiplying the binary mapping y by the parity check matrix as

$$s(y) = H \cdot |bin(y)|. \tag{13}$$

where bin maps values of $[-\infty, \infty]$ to binary values 0 and 1. The function $h(y)$ is then multiplied by the embedding dimension tensor and subsequently fed into the code-aware self-attention module, as illustrated in Fig. 15.

2) Code Aware Self-Attention: In self-attention mechanisms, masks prevent specific positions from attending to others, preserving the integrity of sequential data by maintaining causal or structural relationships. In error detection and correction, a decoder uses parity check equations to analyze

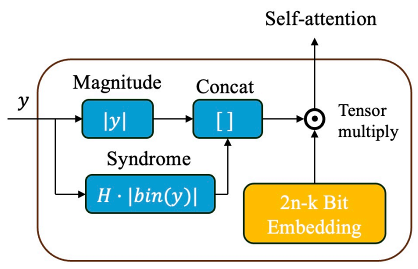


Fig. 15. Pre-processing of ECCT.

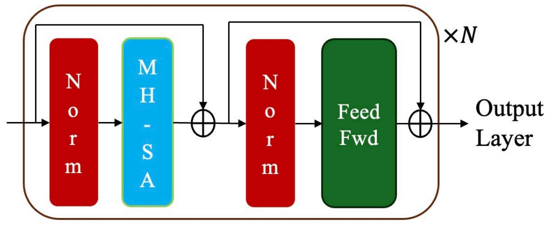


Fig. 16. ECCT Decoder Layer.

and compare received bits, with a non-zero syndrome indicating channel errors. As shown in Fig. 16, one decoder layer comprises two residual networks: one includes multi-head self-attention (MH-SA), and the other contains a normalization function and one NN. The decoder layer iterates N times, where N is the specified number of decoder layers.

As proposed in [14], transformer-based architecture, bit comparisons are facilitated by the self-attention mechanism. However, comparing every pair of elements is unnecessary and slowing training speed since information bits are not interrelated. To address this, the self-attention mask is leveraged to limit self-attention only pairing with relevant bits, ensuring that syndrome values depend only on corresponding parity check bits. An algorithm is introduced to define a mask using the matrix $\mathbf{H}$ within the self-attention mechanism. Figure 17 illustrates the relationship between parity check bits and the syndrome, while Fig. 18 displays the mask result. In the upper left part of the mask, syndrome bits set to 1 will result in only the corresponding parity check bits and message bits being set to 1. The mask is initially set as the identity matrix. It unmasks positions in each row of the matrix $\mathbf{H}$ where there are pairs of ones. This process reflects the pairwise bit relationships that influence the syndrome during decoding. The procedure is summarized in Algorithm 1.

3) Output Layer: The output layer restores the data to its original shape. Initially, the input data undergoes embedding, increasing its dimensionality. To calculate the error rate, reverting the data to its original shape is necessary. This is accomplished by applying two NNs as shown in Fig. 19. The first NN reduces the dimensionality introduced by the multi-head mechanism, while the second NN addresses the dimensionality increase caused by the initial embedding.

4) Post-processing: In the post-processing step, the output is multiplied by $|y|$, which locates the possible error bit

Algorithm 1 Mask Construction Pseudo Code

```
1: function g(H)
2:     $k, n = H$.shape
3:     $z = n - k$
4:     m = eye $(2n - z)$
5: for $x$ in range $(0, n - z)$ do
6:         it = where $(H[x] == 1)$
7:     for $y$ in it do
8:             m $[n + x, y]$ = m $[y, n + x] = 1$
9:         for $z$ in it do
10:                m $[y, z]$ = m $[z, y] = 1$
11:        end for
12:    end for
13: end for
14:    return $-\infty(\neg \text{m})$
```

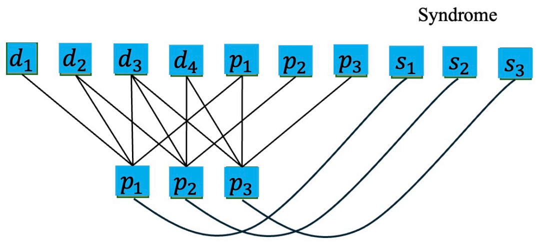


Fig. 17. ECCT Tanner Graph.

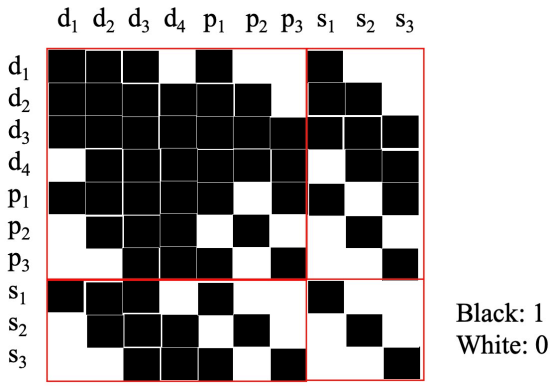


Fig. 18. ECCT Mask.

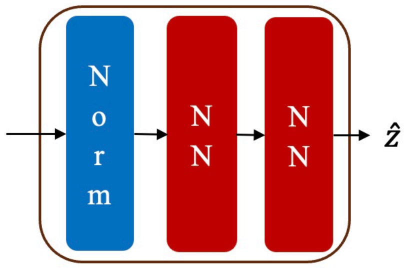


Fig. 19. ECCT Output Layer.

positions and flips. The post-processing operation is defined as follows:

$$\hat{x} = \text{signtobin}(z_{\text{pred}} \cdot \text{sign}(y)), \tag{14}$$

where the signtobin function maps values of -1 and 1 to binary values 0 and 1, the sign$(\cdot)$ function is employed to map input values to either -1 or 1, depending on the sign of the input, with positive inputs resulting in 1 and negative inputs in -1; z_{pred} represents the output of the transformer.

5) Training: The preprocessing step sets the model's embedding dimension to d. The model utilizes 8 head self-attention mechanism with 6 self-attention iterations. The author recommends using the Adam optimizer with 128 samples per minibatch, running for 1000 epochs with 1000 minibatches per epoch. The learning rate is initialized at 10^{-4} and decays to $2 \cdot 10^{-7}$ using a cosine decay scheduler by the end of the training. No warmup or early stopping is employed. Although the performance continuously improves with an increasing number of epochs, the rate of improvement diminishes as the number of epochs increases.

B. Results and Discussion

Results are shown in Figs. 20, 21 and, 22 for BCH $(31, 16)$, BCH $(63, 51)$, and BCH $(127, 64)$ codes, respectively. The model was trained for 4000 epochs for BCH $(63, 51)$, 2000 epochs for BCH $(31, 16)$, 3000 epochs for BCH $(127, 64)$ to achieve the reported performance.

For short-length codewords, such as BCH $(31, 16)$, the ECCT exhibits good performance with potential for improvement. As the number of epochs increases, the learning rate decreases significantly. Although the performance improve, the improvement will be minimal, and the training process will consume a substantial amount of time. However, as the code length increases, such as in BCH $(63, 51)$, the ECCT's performance diminishes.

When compared to the information set decoding (ISD) [24], the ECCT outperforms ISD at $q = 0$ but performs significantly worse than ISD at $q = 1$, where q is the quantization method representing the noise level or error probability. Additionally, ECCT is more complex and requires extensive training time, making it less efficient. For longer codes, such as BCH $(127, 64)$, the ECCT's performance further degrades. Despite increasing the number of self-attention layers (N), the improvement is marginal and the performance is only slightly better than ISD at $q = 0$. With 4000 epochs and over 9 million trainable parameters for BCH $(127, 64)$, ECCT is not a practical choice for such long codes.

Based on the pre-processing and post-processing methods, ECCT employs the magnitude and syndrome vectors to train the multiplicative noise $\tilde{z}_s$, defined as

$$y = x_s + z = x_s \tilde{z}_s. \tag{15}$$

Transformers are highly effective for modeling consecutive time-series data, which typically exhibit strong temporal dependencies. However, the information bits are independent, and parity check bits demonstrate weak interdependency. Thus, the direct application of transformers can lead to bad performance. Specifically, if a transformer model is trained to

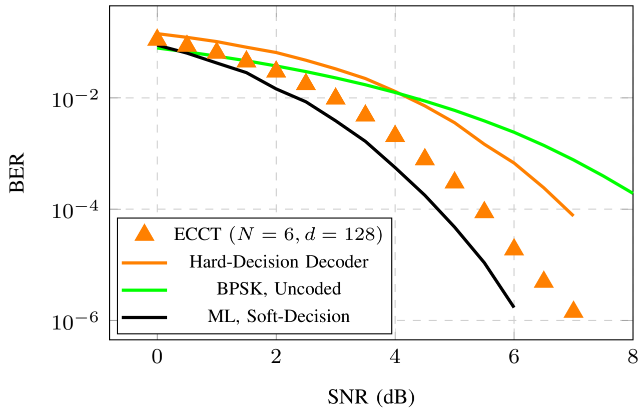


Fig. 20. BCH $(31, 16)$: SNR vs. BER to compare the performance of the ECCT.

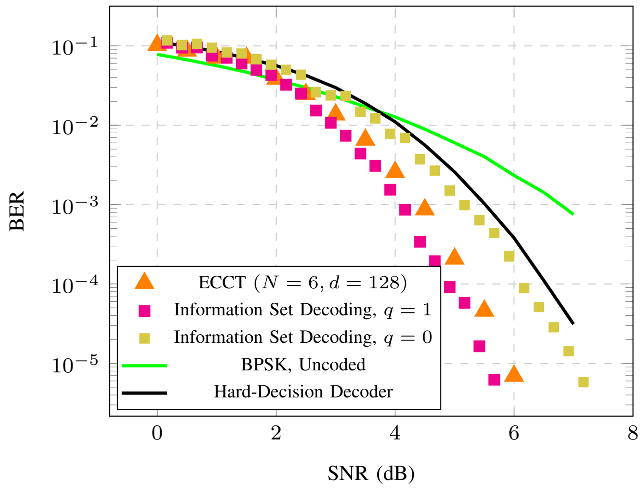


Fig. 21. BCH $(63, 51)$: SNR vs. BER to compare the performance of the ECCT.

learn the relationships between information bits and parity bits, it risks overfitting to noise within the training data, thereby degrading performance. To address this, the author designed ECCT to learn multiplicative noise, enabling it to mitigate the noise effects on the signal. This approach is particularly advantageous in low SNR scenarios.

VI. Cross-attention Message Passing Transformer

CrossMPT, which is designed based on cross-attention transformer represents the state-of-the-art in error correction transformers [15].

A. Architecture and Training

Inspired by BP, CrossMPT facilitates message exchanges between the magnitude $|y|$ and the syndrome $s(y)$. Initially, $|y|$ is updated using a masked cross-attention block, with $|y|$ as the query and $s(y)$ as both the key and the value. The resulting attention map of dimensions $n \times (n - k)$ models the “magnitude-syndrome” relation, using the transpose of

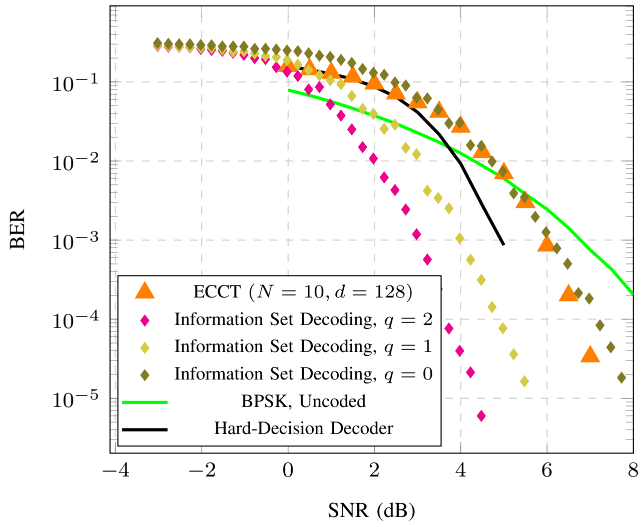


Fig. 22. BCH $(127, 64)$: SNR vs. BER to compare the performance of the ECCT.

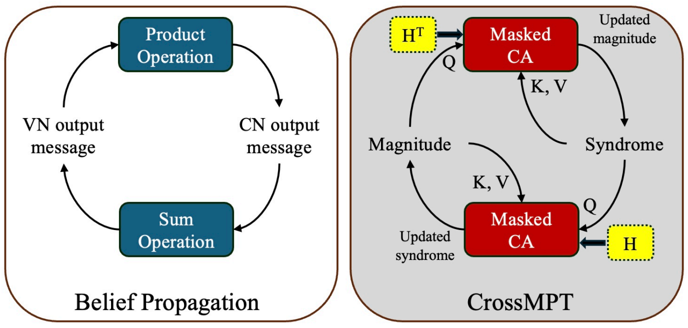


Fig. 23. Comparison of BP and CrossMPT.

the parity-check matrix H as the mask. This update yields a refined magnitude vector. Subsequently, $s(y)$ is updated in another masked cross-attention block, where $s(y)$ serves as the query, and the updated $|y|$ acts as both the key and the value, with H being the mask. This process iteratively refines both the magnitude and the syndrome to better identify the multiplicative noise. Fig. 23 compares the sum-product algorithm and CrossMPT as message-passing approaches. In the sum-product algorithm, variable and check node messages are updated through sum and product operations. Similarly, CrossMPT iteratively updates magnitude and syndrome vectors using masked cross-attention blocks.

1) Input Embedding: Unlike ECCT, which concatenates the magnitude and the syndrome in the pre-processing step, CrossMPT embeds the magnitude and the syndrome separately and inputs them into the cross-attention layer, illustrated in Fig. 24.

2) Cross Attention: In the decoder layer, illustrated in Fig. 24, the first cross-attention query originates from the normalized embedded magnitude, while the key and the value are derived from the syndrome. After the initial cross-attention and FFNN, the output serves as the key and the value for the

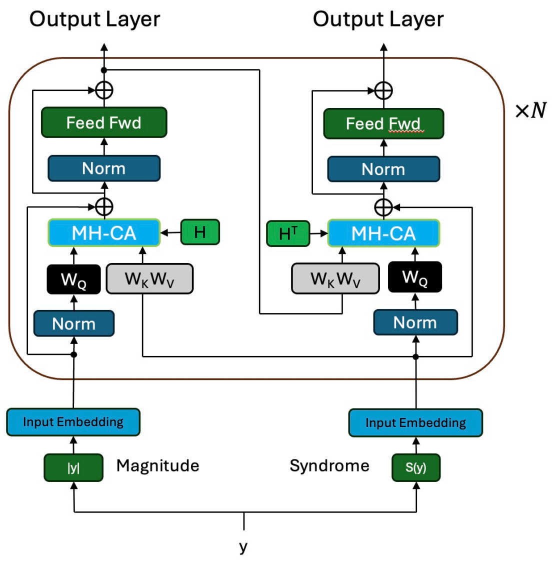


Fig. 24. CrossMPT Decoder Layer.

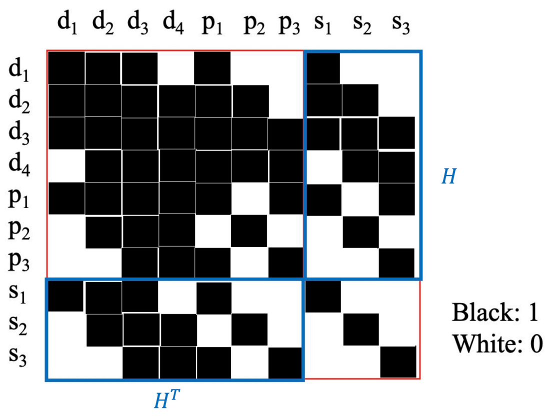


Fig. 25. CrsossMPT Mask: Blue squares indicates CrossMPT Mask of H and H^T.

next cross-attention. The query for this stage comes from the normalized syndrome. Unlike ECCT, which uses an algorithm to create a mask, CrossMPT directly employs $\mathbf{H}$ and $\mathbf{H^T}$ matrices as masks as shown in Fig. 25. This method obviates the need for individual comparisons of all queries and keys, thereby enhancing the training speed.

3) Output Layer: The two outputs are concatenated before the output layer. Later, similar to the ECCT output layer, two NNs are used to decrease the high dimensionality caused by the multi-head mechanism and embedding.

4) Training: CrossMPT uses Adam optimizer and the authors suggest 1000 epochs. Each epoch consists of 1000 minibatches, where each minibatch is composed of 128 samples.

B. Results and Discussion

For the BCH $(31, 16)$ code, CrossMPT demonstrates a performance improvement of 0.25 dB over ECCT, as shown in Fig. 27. However, for the BCH $(63, 51)$ code, the performance difference between ECCT and CrossMPT is minimal as illustrated in Fig. 28. Despite this improvement, CrossMPT does not reach ML performance for the BCH $(31, 16)$ code. For the BCH $(63, 51)$ code, CrossMPT's performance is nearly identical to ECCT, even after 4000 epochs with the same number of decoder layer iterations and embedding dimensions. CrossMPT still falls short of outperforming ISD $q = 1$ for the BCH $(63, 51)$ code. Although CrossMPT achieves marginally better performance with extended training time, its high model complexity and over 1 million trainable parameters render it unsuitable for ECCs, similar to ECCT. The substantial size of the model demands prolonged training time and extensive resource consumption.

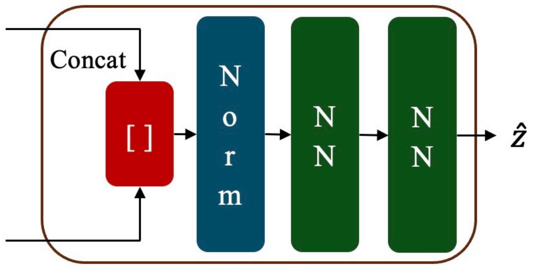


Fig. 26. CrossMPT Output Layer.

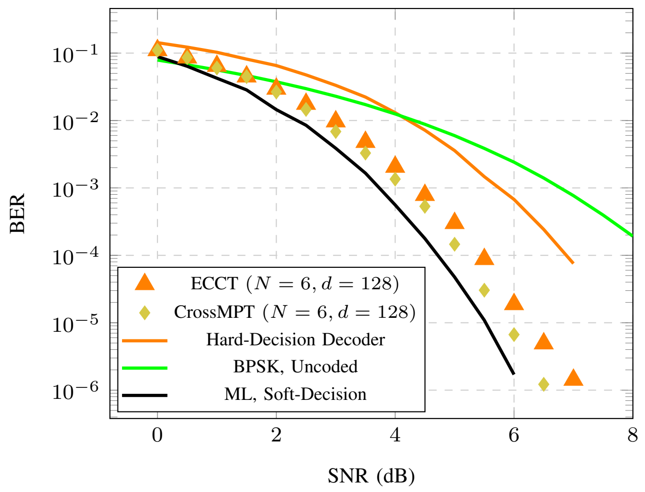


Fig. 27. BCH $(31, 16)$: SNR vs. BER to compare the performance of the CrossMPT.

VII. Conclusion and Future Prospects

This work evaluates the performance of neural network-based decoders for decoding FEC codes, focusing on the SLNN and MLNN architectures. First, the SLNN model that was recently-proposed in the literature was demonstrated to have BER and BLER performance comparable to SDML decoding for the Hamming $(7, 4)$ and Parity $(26, 10)$ codes. Then, our analysis revealed that the hidden layer in SLNN could be omitted without performance loss, simplifying the model and the training. However, even after this simplification, implementing an SLNN for longer codes was shown to be

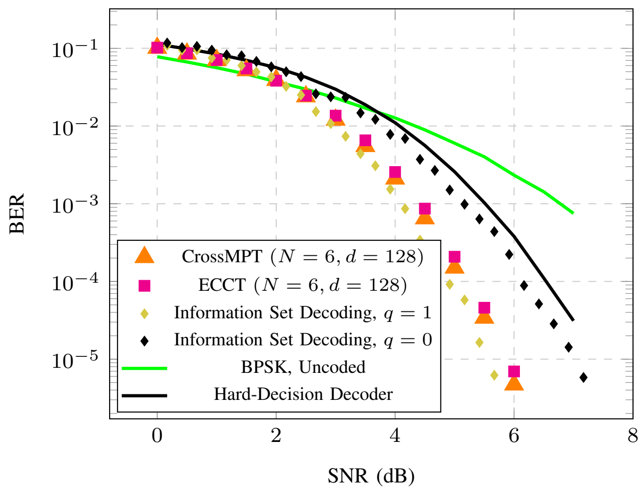


Fig. 28. BCH $(63, 51)$: SNR vs. BER to compare the performance of the CrossMPT.

impractical due to the exponential dependence between the code length and the size of the output layer.

To solve this problem, the MLNN architecture that was also recently-proposed in the literature was studied. The BER performance of the MLNN architecture matched to that of SDML decoding with much complex structure. We then proposed an optimized MLNN architecture with a hidden layer size scaling exponentially with the code length and with softmax activation function. Our modified MLNN architecture achieves BER performance of the original MLNN architecture while having reduced complexity.

Finally, we studied recently-proposed ECCT and CrossMPT architectures for decoding FEC codes for the BCH $(31, 16)$, BCH $(63, 51)$, and BCH $(127, 64)$ code. We show that similar to SLNN and MLNN, ECCT and CrossMPT are also computationally infeasible to implement for decoding longer codes with prolonged training times and resource consumption. Moreover, their performance stayed worse than the well-known and low-complexity information set decoding approach. The transformers' approach of learning multiplicative noise, while capable of identifying relationships between coded bits, ultimately resulted in low efficiency and performance, indicating that applying transformers to decode FEC codes is not a suitable choice and BP-based decoders are still the best option available.

VIII. Acknowledgements

The author would like to thank Prof. Alex Alvarado for the opportunity to pursue this master's graduation project, as well as for his invaluable guidance and insightful discussions throughout this research. Special thanks to Peter Scheepers for his supports in the beginning of the project. The author also thanks Dr. Yunus Can Gültekin for proofreading and improving the paper.

References

[1] Richard Wesley Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.
[2] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.
[3] Marcel J. E. Golay. Notes on digital coding. *Proc. IRE*, 37:657, June 1949.
[4] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, July 2009.
[5] A. Vardy and Y. Be'ery. Maximum-likelihood soft decision decoding of bch codes. *IEEE Transactions on Information Theory*, 40(2):546–554, March 1994.
[6] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February 2001.
[7] S. E. El-Khamy, E. A. Youssef, and H. M. Abdou. Soft decision decoding of block codes using artificial neural network. In *Proceedings IEEE Symposium on Computers and Communications*, pages 234–240, 1995.
[8] J. Kuck, S. Chakraborty, H. Tang, R. Luo, J. Song, A. Sabharwal, and S. Ermon. Belief propagation neural networks, 2020.
[9] Sietsma and Dow. Neural net pruning-why and how. In *IEEE 1988 International Conference on Neural Networks*, volume 1, pages 325–333, 1988.
[10] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *CoRR*, abs/2003.03033, 2020.
[11] Jeremy Wang, Wohlwend, and Tao Lei. Structured pruning of large language models, 2019.
[12] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.
[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
[14] Yoni Choukroun and Lior Wolf. Error correction code transformer, 2022.
[15] Seong-Joon Park, Hee-Youl Kwak, Sang-Hyo Kim, Yongjune Kim, and Jong-Seon No. Crossmpt: Cross-attention message-passing transformer for error correcting codes, 2024.
[16] D.E. Hocevar. A reduced complexity decoder architecture via layered decoding of ldpc codes. In *IEEE Workshop onSignal Processing Systems, 2004. SIPS 2004.*, pages 107–112, 2004.
[17] Danil Prokhorov. *Neural Networks in Automotive Applications*, pages 101–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
[18] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.
[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
[20] Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. On deep learning-based channel decoding, 2017.
[21] Sudarshan Adiga, Xin Xiao, Ravi Tandon, Bane Vasić, and Tamal Bose. Generalization bounds for neural belief propagation decoders. *IEEE Transactions on Information Theory*, 70(6):4280–4296, June 2024.
[22] Eliya Nachmani, Yair Beery, and David Burshtein. Learning to decode linear codes using deep learning, 2016.
[23] C. T. Leung, R. V. Bhat, and M. Motani. Low latency energy-efficient neural decoders for block codes. *IEEE Transactions on Green Communications and Networking*, 7(2):680–691, June 2023.
[24] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 69–89, Cham, 2017. Springer International Publishing.