Received 12 November 2024; revised 9 December 2024; accepted 28 December 2024. Date of publication 31 December 2024;
date of current version 16 January 2025.

Digital Object Identifier 10.1109/OJCOMS.2024.3524429

Iterative Syndrome-Based Deep Neural Network Decoding

DMITRY ARTEMASOV (Graduate Student Member, IEEE), KIRILL ANDREEV (Member, IEEE), PAVEL RYBIN (Member, IEEE), AND ALEXEY FROLOV (Member, IEEE)

Center for Next Generation Wireless and IoT, Skolkovo Institute of Science and Technology, 121205 Moscow, Russia

CORRESPONDING AUTHOR: D. ARTEMASOV (e-mail: d.artemasov@skoltech.ru)

This work was supported by the Russian Science Foundation under Project 23-11-00340 (https://rscf.ru/en/project/23-11-00340/).

This work was presented in part at the GLOBECOM 2023 conference [1] [DOI: 10.1109/GCWkshps58843.2023.10465120].

ABSTRACT While the application of deep neural networks (DNNs) for channel decoding is a well-researched topic, most studies focus on hard output decoding, potentially restricting the practical application of such decoders in real communication systems. Modern receivers require iterative decoders, a pivotal criterion for which is the ability to produce soft output. In this paper, we focus on this property. We begin by modifying the syndrome-based DNN-decoding approach proposed by Bennatan et al. (2018). The DNN model is trained to provide soft output and replicate the maximum a posteriori probability decoder. To assess the quality of the proposed decoder's soft output, we examine the iterative decoding method, specifically the turbo product code (TPC) with extended BCH (eBCH) codes as its component codes. A sequential training procedure for optimizing the behavior of component decoders is utilized. We illustrate that the described approach achieves exceptional performance results and is applicable for iterative codes with larger code lengths $[n = 4096, k = 2025]$, compared to state-of-the-art DNN-based methods. Finally, we address the issues of computational complexity and memory requirements of DNN-based decoding by analyzing the model's compression limits through pruning and matrix decomposition methods.

INDEX TERMS Channel decoding, complexity reduction, deep neural networks, iterative codes, soft-output.

I. INTRODUCTION

THE FIELD of machine learning (ML) has experienced rapid expansion in recent years, with the application of deep neural networks (DNNs) becoming increasingly prevalent. Over the past decade, DNNs have driven remarkable progress in areas such as text processing, computer vision, and speech recognition [2], all of which involve natural signals. However, the application of machine learning techniques in the field of communications remains relatively underexplored. This paper examines the use of ML methods to address the problem of channel decoding. In this context, decoding can be viewed as a classification problem: the goal is to assign the channel output to one of a possible classes (codewords). The primary distinction from traditional classification tasks lies in the exponential growth of these classes as the message length increases.

The concept of utilizing DNNs for channel decoding has been previously explored in earlier works like [3], [4]. However, these methods were largely set aside due to the computational limitations of that time. The approach was revisited in a more recent work [5], which examined decoding in a binary-input channel affected by additive white Gaussian noise (AWGN) and proposed a fully connected neural network for this purpose. While a common challenge in ML is the collection and labeling of datasets, this issue is easily addressed in decoding by generating datasets through sampling codewords and applying random AWGN. Despite this, the model in [5] encountered the "curse of dimensionality", due to the exponential growth in the number of codewords with the increase in information message length. Training of such a network on all possible codewords becomes impractical for realistic parameters. The expectation

was that the network can learn the code structure from a subset of codewords. However, the main findings in [5] revealed that fully connected networks fail to effectively capture the code structure. As a result, this approach was deemed suitable only for very short codes.

Another research field focuses on augmenting classical decoding algorithms with trainable weights. In what follows, this methodology is referred to as the model-based approach. Several studies [6], [7], [8], [9], [10], [11], [12] have explored the belief propagation (BP) algorithm, including Sum-Product and Min-Sum variants, which is applicable to any linear code but performs best for sparse-graph codes, such as low-density parity-check (LDPC) codes [13]. The idea is to unwrap (or unroll) the underlying Tanner graph and obtain a sparse NN, which repeats the decoder operations but is equipped with trainable parameters. This technique has enhanced the performance of decoding BCH codes [6], [7], [11] and LDPC codes [8], [9], [14]. A further development introduced the use of hyper-networks, which replaced conventional activation functions with trainable ones [15], [16]. Cammerer et al. later enhanced this method by introducing trainable functions to replace the message updates at the nodes and edges. This enhancement allowed the neural network to learn a more generalized version of the message-passing algorithm [17].

An alternative approach, proposed in [18], considers a syndrome-based decoding algorithm applicable to any linear code. The general syndrome decoding involves the use of a mapping (syndrome to the coset leader), which has an exponential size relative to the number of parity-check bits. In [18], the authors propose approximating this table using a neural network. We note that the syndrome does not depend on the codeword and, therefore, we do not require the NN to have a special structure. Yet, the best results were achieved using recurrent neural networks (RNNs) [18]. More recently, the syndrome-based approach has been adapted to transformer and denoising diffusion architectures [19], [20]. For further details and a comprehensive review of advances in deep learning-based decoding, we refer readers to [21].

Existing DNN-based decoders exhibit three main problems: (i) the majority of architectures are designed to produce hard output; (ii) their application is restricted to short codes due to the "curse of dimensionality" problem; (iii) excessive complexity limits their application in the real-world communication systems. Let us consider these problems in more detail. Hard-output decoding requires the receiver to return the estimated information word as a bit sequence. Modern receivers, such as multiple-input multiple-output (MIMO) systems [22], and modern codes composed of short component codes [23], often rely on iterative (or turbo) decoders. In these systems, soft-output decoding is crucial. While some works (e.g., [17], [18]) mention the potential for obtaining soft-outputs using the proposed DNN architectures, to the best of our knowledge, the quality of such outputs has not been thoroughly explored in the literature. Let us consider the second issue. Training a neural network to generalize over a long code is often challenging due to factors like the number of parameters, training stability, and the time required for training. Recent research has focused on leveraging the autoencoder (AE) architecture within turbo decoding schemes [24], [25]. Turbo autoencoders involve jointly training linear or convolutional neural network (CNN) layers for both encoding and iterative decoding tasks. It's worth noting that this approach requires training component decoders instead of generalizing the entire turbo code structure. However, the decoder architecture remains similar to one outlined in [5] and thus may suffer from the "curse of dimensionality". Regarding the third issue raised, we highlight the fact that DNN-based decoders built upon general neural architectures are typically over-parameterized, leading to redundancy and wastage of both computational and memory resources [26], which limits their application in real systems' hardware.

In this paper, we address the issues mentioned above. In what follows we focus on the syndrome-based approach [18] and do not consider DNN decoders designed as an augmentation of Tanner-based decoders [8]. The reason is that we deal with short component codes. It is known that short, sparse codes perform poorly in comparison to polar and BCH codes. At the same time, the application of Tanner-based decoders is limited to sparse codes only.

Our contribution is as follows:

- We extend the syndrome-based decoding approach of [18] to provide soft output. The major modification is in the training process and the loss function, which now incorporates a maximum a posteriori probability (MAP) output-based regularization term to enhance the soft output quality.
- To evaluate the quality of the soft output we consider an iterative decoding scheme: the turbo product code (TPC) with [64, 45] extended BCH (eBCH) as row and column component codes. We employ a sequential training strategy, enabling precise tuning of each component decoder to the output of the preceding one. We demonstrate that the performance of the TPC with the proposed method for decoding component codes is very close to that of TPC with component MAP decoding, while significantly outperforming TPC with Chase decoding [27] combined with the soft output calculation method [28].
- Finally, we address the complexity and memory requirements of DNN-based decoders. Specifically, we explore the limits of model compression using pruning and low-rank approximation methods.

Fig. 1 depicts the schematic representation of the proposed soft-output decoding method.

The paper is organized as follows. In Section II, the system model is introduced. In Section III, we elaborate on the modifications made to the syndrome-based approach to enable soft-output. This section outlines the model architecture, discusses the regularization functions employed to

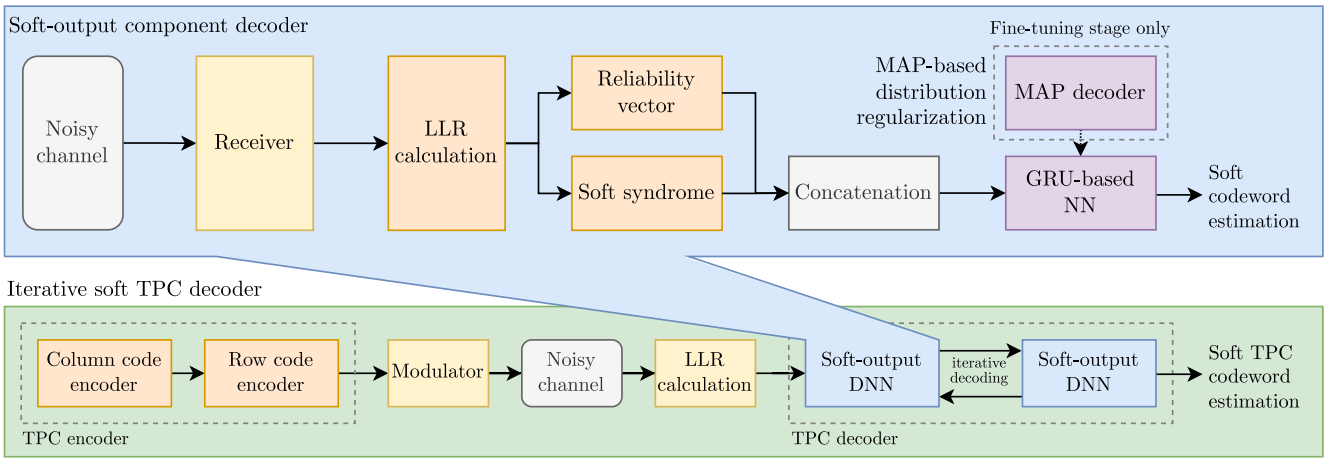


FIGURE 1. Schematic representation of the developed iterative syndrome-based DNN decoder.

enhance soft-output quality, and presents numerical results. Section IV considers the application of the developed decoder in the turbo product scheme, detailing the sequential training procedure and providing TPC decoding performance results. Section V addresses the issue of DNN-based decoder complexity and memory requirements through pruning and low-rank approximation. Section VI summarizes the paper and discusses the achieved results.

II. SYSTEM MODEL

The goal of the user is to transmit a k-bit information sequence $\mathbf{u} \in \{0,1\}^k$. The system employs a binary linear block code $\mathcal{C}$ with length n and dimension k. The parity-check matrix (PCM) and generator matrix of the code $\mathcal{C}$ are denoted by $\mathbf{H}$ and $\mathbf{G}$, respectively. The information sequence $\mathbf{u}$ is encoded into a codeword $\mathbf{c} = (c_1, \ldots, c_n) = \mathbf{uG} \in \{0,1\}^n$. Following this, the modulation is applied. In this paper, we utilize binary phase-shift keying (BPSK)

$$x_i = \begin{cases} 1, & \text{if} \quad c_i = 0 \\ -1, & \text{if} \quad c_i = 1. \end{cases}, \forall i \in [1, \ldots, n]. \tag{1}$$

We consider transmission over a binary-input AWGN channel. The receiver observes the signal:

$$\mathbf{y} = \mathbf{x} + \mathbf{z}, \tag{2}$$

where $\mathbf{z}$ represents noise following a normal distribution $\mathcal{N}(0, \sigma^2 \mathbf{I}_n)$, with $\mathbf{I}_n$ being the $n \times n$ identity matrix. In this context, the signal-to-noise ratio (SNR) is defined as $E_s/N_0 = 1/2\sigma^2$.

As usual [29], the decoder input is represented by a vector of log-likelihood ratios (LLRs) $\boldsymbol{\gamma} \in \mathbb{R}^n$. Each element of which is given by:

$$\gamma_i = \log \frac{p(y_i|c_i = 0)}{p(y_i|c_i = 1)} = \frac{2y_i}{\sigma^2}, \quad \forall i \in [1, \ldots, n], \tag{3}$$

where $\log(\cdot)$ denotes the natural logarithm, and the probability density function of a random variable following the distribution $\mathcal{N}(x_i, \sigma^2)$ is defined as $p(y_i|c_i) = 1/\sqrt{2\pi\sigma^2} \exp\left[-(y_i - x_i)^2/(2\sigma^2)\right]$.

Now, let us describe the decoding performance metric. We begin by considering hard-output decoding. Let $\hat{\mathbf{u}} = (\hat{u}_1, \ldots, \hat{u}_k) \in \{0,1\}^k$ be the estimated information word. The performance is measured using the bit error rate (BER), given by $P_b = \frac{1}{k} \sum_{i=1}^{k} \Pr[u_i \neq \hat{u}_i]$.

To assess soft output quality, we compare the decoder output to the bit-wise MAP output $\boldsymbol{\gamma}^* = (\gamma_1^*, \ldots, \gamma_n^*)$

$$\gamma_i^* = \log \frac{\Pr[c_i = 0|\mathbf{y}]}{\Pr[c_i = 1|\mathbf{y}]} = \log \frac{\sum\limits_{\mathbf{c} \in \mathcal{C}, c_i = 0} \exp\left[(\mathbf{1} - \mathbf{c})\boldsymbol{\gamma}^T\right]}{\sum\limits_{\mathbf{c} \in \mathcal{C}, c_i = 1} \exp\left[(\mathbf{1} - \mathbf{c})\boldsymbol{\gamma}^T\right]}, \quad \forall i \in [1, \ldots, n]. \tag{4}$$

Here, $\mathbf{1}$ represents the all-ones vector, and $\boldsymbol{\gamma}^T$ is the transposed LLR vector. For a detailed derivation, we refer the reader to [30].

III. SOFT-INPUT SOFT-OUTPUT DNN-BASED DECODING

A. SYNDROME-BASED APPROACH

Syndrome decoding is a classical method for decoding linear codes. It involves creating a standard array, where each syndrome corresponds to the most likely error vector, known as the coset leader [31]. The paper of Bennatan et al. [18] extends the classical syndrome decoding and proposes to utilize both the reliability vector and the syndrome for implicit neural network-based standard array reconstruction. The performance of the decoder described in [18] is validated for hard output decoding only. In this work, we suggest modifying the pre- and post-processing stages to adapt the framework for soft-output decoding in the AWGN channel.

In [18], the authors suggest using the vector $[|\mathbf{y}|, \mathbf{s}]$ as input of the noise estimator, where $[\cdot, \cdot]$ represents concatenation, $|\mathbf{y}|$ is the reliability vector, and $\mathbf{s} = \text{bin}(\mathbf{y})\mathbf{H}^T$ is the binary syndrome. In such notation $\text{bin}(\cdot)$ implies the following mapping:

$$\text{bin}(y_i) = \begin{cases} 0, & \text{if} \quad y_i \geq 0 \\ 1, & \text{if} \quad y_i < 0. \end{cases}, \forall i \in [1, \ldots, n] \tag{5}$$

Algorithm 1 Soft-Output Syndrome-Based DNN Decoding

Input: $\boldsymbol{\gamma} \in \mathbb{R}^n$ – input LLRs, $\tilde{\mathbf{s}} \in \mathbb{R}^{n-k}$ – soft syndrome
Output: $\hat{\boldsymbol{\gamma}} \in \mathbb{R}^n$ – transmitted message LLRs estimation
1: $\hat{\mathbf{z}} \leftarrow \mathcal{F}([|\boldsymbol{\gamma}|, \tilde{\mathbf{s}}])$
2: $\hat{\boldsymbol{\gamma}} \leftarrow \boldsymbol{\gamma} - \text{sign}(\boldsymbol{\gamma}) \odot \hat{\mathbf{z}}$
3: **return** $\hat{\boldsymbol{\gamma}}$

To bypass the hard-decision stage in preprocessing, we propose using the so-called *soft syndrome* introduced by Lugosch and Gross [32]. Since operations $(\{0, 1\}, \oplus)$ and $(\{1, -1\}, *)$ are isomorphic, the general hard syndrome calculation can be expressed as

$$s_i = \prod_{j \in \mathcal{M}(i)} \text{sign}(y_j), \forall i \in [1, \ldots, n-k], \tag{6}$$

where $\text{sign}(y_j)$ denotes the sign of y_j, and $\mathcal{M}(i)$ represents the set of parity check positions in the i-th row of $\mathbf{H}$.

Therefore, the hard syndrome relaxation (soft syndrome) for the decoder input LLR vector $\boldsymbol{\gamma}$ can be presented as

$$\tilde{s}_i = \min_{j \in \mathcal{M}(i)} |\gamma_j| \prod_{j \in \mathcal{M}(i)} \text{sign}(\gamma_j), \forall i \in [1, \ldots, n-k]. \tag{7}$$

In this notation, $|\gamma_j|$ denotes the absolute value of the LLR γ_j. In the description that follows, the vector $\mathbf{d} = [|\boldsymbol{\gamma}|, \tilde{\mathbf{s}}] \in \mathbb{R}^{2n-k}$ represents the input of the noise estimator.

The proposed decoding method is outlined in Algorithm 1, where the symbol $\odot$ denotes the Hadamard product. In the following, we designate the noise estimation operator by $\mathcal{F}(\cdot)$ and choose to use the DNN model to implement it.

B. DNN-BASED NOISE ESTIMATOR

The primary objective of the neural network $\mathcal{F}(\cdot)$ is to estimate the noise vector. The selection of the optimal DNN model architecture remains an open question. In this paper we compare the hard decoding performance of the stacked gated recurrent unit (Stacked-GRU) model [18] with transformer (ECCT) [19] and denoising diffusion (DDECC) [20] for various models' parameters: number of layers L and the dimension of the embedding d_e (see Fig. 2).[1] Considering both the decoding performance and the training time of various architectures, we choose a Stacked-GRU for the soft-output decoding task. The complete description of the Stacked-GRU architecture is provided in the Appendix A.

During model training, the loss is derived from the output of the DNN-based decoder and the binary codeword. A binary cross-entropy (BCE) loss function $\mathcal{L}_{BCE}(\hat{\boldsymbol{\gamma}}, \mathbf{c})$ is applied for this purpose.

$$\mathcal{L}_{BCE}(\hat{\boldsymbol{\gamma}}, \mathbf{c}) = -\frac{1}{n} \sum_{i=1}^{n} c_i \log \sigma(-\hat{\gamma}_i) + (1 - c_i) \log(1 - \sigma(-\hat{\gamma}_i)). \tag{8}$$

Here $\sigma(\cdot)$ denotes the sigmoid function.

[1]Models from [19] and [20] are trained for 10^4 epochs with default parameters specified in the original repositories.

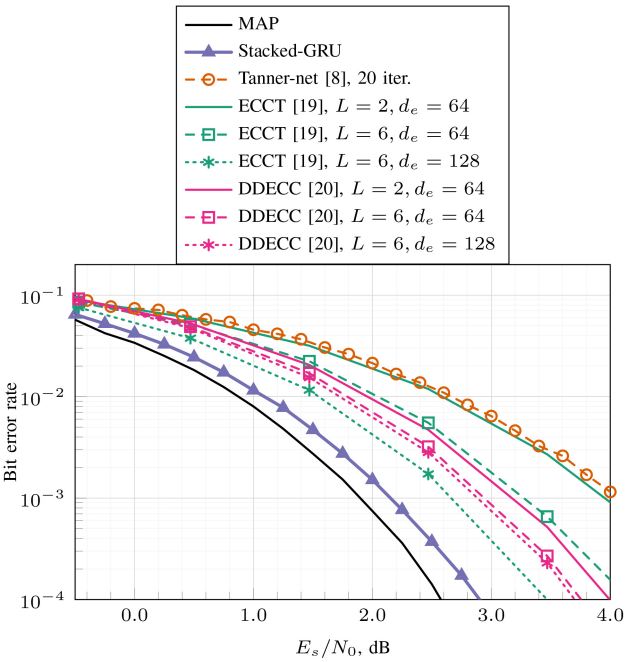


FIGURE 2. Bit error rate results of various DNN decoders for [64, 45] eBCH code.

C. SOFT-OUTPUT QUALITY OPTIMIZATION

To optimize the soft-output distribution, we suggest adding a regularization term to the loss function during the final epochs of model training. We propose two regularization methods: Mean Squared Error (MSE) and a moments-based approach. The MAP decoder outputs, represented as LLRs $\boldsymbol{\gamma}^*$ (4), are used as the reference.

The MSE regularization, denoted as $\mathcal{L}_{MSE}(\hat{\boldsymbol{\gamma}}, \boldsymbol{\gamma}^*)$, is formulated in a pointwise manner, compelling the proposed decoder to replicate the exact output of the MAP decoder. The moments-based regularization $\mathcal{L}_M(\hat{\boldsymbol{\gamma}}, \boldsymbol{\gamma}^*)$ minimizes the differences in the expectation and variance between the distributions produced by the MAP decoder and the DNN-based decoder. This is achieved using a weighted sum of the squared errors for the first and second moments, with the weighting coefficient $\rho_M \in [0, 1]$ balancing their contributions.

$$\mathcal{L}_{\text{MSE}}(\hat{\boldsymbol{\gamma}}, \boldsymbol{\gamma}^*) = \frac{1}{n} \sum_{i=1}^{n} (\hat{\gamma}_i - \gamma_i^*)^2 \tag{9}$$

$$\mathcal{L}_{\text{M}}(\hat{\boldsymbol{\gamma}}, \boldsymbol{\gamma}^*) = \rho_{\text{M}} \Big(\mathbb{E}(|\hat{\boldsymbol{\gamma}}|) - \mathbb{E}(|\boldsymbol{\gamma}^*|) \Big)^2 + (1 - \rho_{\text{M}}) \Big(\text{Var}(|\hat{\boldsymbol{\gamma}}|) - \text{Var}(|\boldsymbol{\gamma}^*|) \Big)^2 \tag{10}$$

The resulting loss function with the introduced regularization term is defined as

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \alpha_{\text{Reg}} \mathcal{L}_{\text{Reg}}. \tag{11}$$

Here, $\mathcal{L}_{Reg}$ represents the output of the selected regularization function, and α_{Reg} denotes its scale.

Table 1 summarizes the results of the soft-output optimization with proposed regularization functions.

TABLE 1. Evaluation of the NN-decoder soft-output distribution compared to map output with proposed regularizations.

	$\mathcal{L}_{\text{BCE}}$	$\mathcal{L}_{\text{BCE}} + \alpha_{\text{MSE}}\mathcal{L}_{\text{MSE}}$	$\mathcal{L}_{\text{BCE}} + \alpha_{\text{M}}\mathcal{L}_{\text{M}}$
KL divergence (12):	0.0178	0.0073	0.0094
Pointwise MSE (9):	4.906	3.547	4.864
Squared error of expectations (10):	0.606	0.148	0.018
Squared error of variances (10):	11.752	7.170	0.231

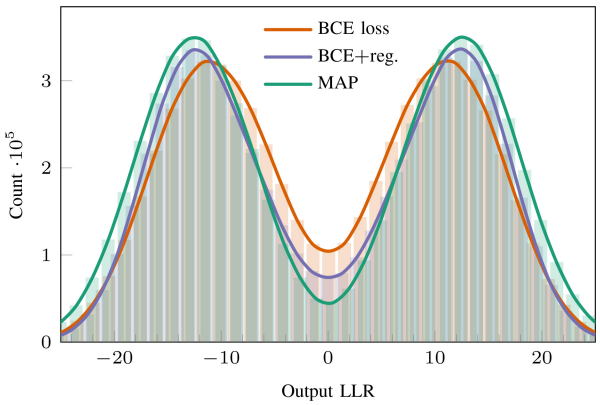


FIGURE 3. Output LLR distributions histogram for $E_s/N_0 = 1$ dB [64, 45] eBCH code. The moments-based approach is used for regularization.

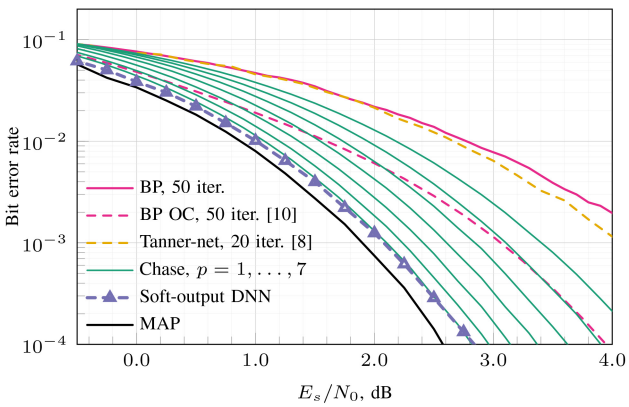


FIGURE 4. BER results for [64, 45] eBCH code.

Columns denote the used loss function for training and rows denote the type of distributions similarity metrics. To calculate the KL divergence, output samples from both the DNN-based decoder and the MAP decoder are accumulated. The probability mass functions for both outputs are calculated using the same set of uniformly distributed values (bins), $\mathcal{X}$. The number of values, $|\mathcal{X}| = 101$, and their range, $[-40, 40]$, are determined empirically to fit the distributions. The KL divergence is computed from the values and associated probabilities from the MAP decoder $P(x)$ and the DNN-based decoder $Q(x)$, as defined in the equation below [33], [34]

$$D_{\text{KL}}(P\|Q) \triangleq \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)}. \tag{12}$$

Fig. 3 depicts the output distributions of the neural decoder trained with BCE loss only, the neural decoder trained with the moments-based regularization, and the MAP decoder, as an example. Results are provided for the decoder trained on [64, 45] eBCH code with evaluation on $E_s/N_0 = 1$ dB. Scales for regularization terms are obtained empirically: $\alpha_{\text{MSE}} = 0.01$, $\alpha_{\text{M}} = 0.1$, $\rho_{\text{M}} = 0.95$.

D. SIMULATION RESULTS

To assess the decoding performance of the proposed framework, we train a Stacked-GRU model with 4 layers, 5 time steps, and a hidden size of $5n$, where n is the code length, on zero codewords using a batch size of 2^{13}. The Adam optimizer [35] is utilized for training, initialized with a learning rate of 10^{-3}. The learning rate is progressively decreased to 10^{-6} by the "reduce on plateau" scheduler. The initial training is performed with the BCE loss (8) only. The MAP-based regularization terms are included for the final epochs due to the complexity of MAP decoding.[2] During training, AWGN is sampled with a uniformly distributed variance, corresponding to the SNR range where each specific code is expected to produce a BER between $5\cdot10^{-2}$ and 10^{-4}. For the [64, 45] eBCH and [64, 45] CRC-Aided (CA)-Polar codes, a training range of $E_s/N_0 = [0, 3]$ dB is selected. For the [64, 21] CA-Polar code, the training range is set to $E_s/N_0 = [-3.5, -0.5]$ dB.

The performance of the soft-output DNN decoder is evaluated in comparison to the Chase decoder (with the number of least reliable positions $p = 1, \ldots, 7$), regular belief propagation with 50 decoding iterations (BP, 50 iter.), belief propagation decoding with overcomplete parity check matrix (BP OC, 50 iter.) [10], and NN-Tanner [8] with 20 decoding iterations for the [64, 45] eBCH code. Successive Cancellation List (SCL) decoder results are used for comparison of CA-Polar codes decoding performance (list size $L_p = 2^1, \ldots, 2^7$ for [64, 45] CA-Polar and $L_p = 2^1, \ldots, 2^5$ for [64, 21] CA-Polar codes). Results are depicted in Fig. 4–6. The Chase and SCL curves are arranged from right to left as the list sizes increase.

From Fig. 4 we can observe that on the [65, 45] eBCH code the Tanner-based decoders (BP and Tanner-net) show

[2]It is worth noting that the proposed soft-output DNN can be utilized without the joint DNN-MAP fine-tuning stage if complexity constraints exist. However, in this case, the DNN output distribution may not align with MAP.

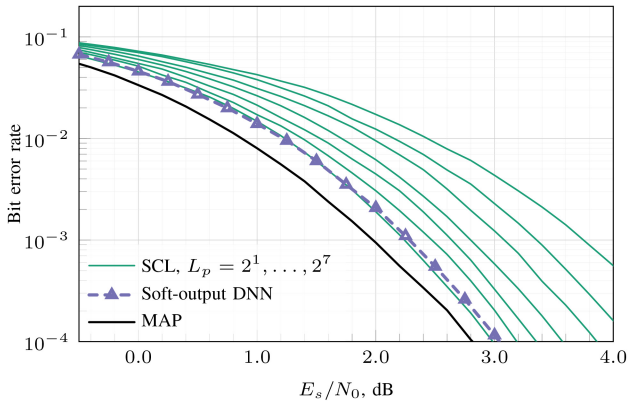


FIGURE 5. BER results for [64, 45] CA-Polar code.

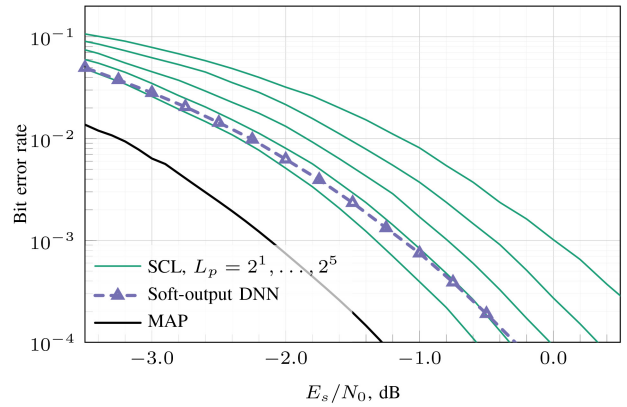


FIGURE 6. BER results for [64, 21] CA-Polar code.

worse performance than other decoding methods. As mentioned in the introduction, such behavior is caused by the nature of the message-passing algorithm, which is more applicable for sparse codes. Also, from Fig. 4 we can observe that the proposed soft-output DNN decoder outperforms the widely used in practice Chase decoder with the number of least reliable positions $p = 7$ and attains MAP decoding performance. On 10^{-3} BER level the gap between MAP and the proposed DNN-based decoder is $\approx$ 0.25 dB. Similar trends are observed for the $[64, 45]$ CA-Polar code (see Fig. 5). The DNN-based decoder provides performance comparable to the SCL decoder with the list size $L_p = 2^7$ and attains MAP performance on 10^{-3} BER level by $\approx$ 0.35 dB. For the lower code rate $[64, 21]$ CA-Polar code, the BER curve of the DNN-based decoder falls between the SCL decoder curves with list sizes $L_p = 2^4$ and $L_p = 2^5$ (see Fig. 6). The worse error correcting capability of the DNN-based decoder on lower code rates is caused by the increase of the syndrome space size. However, the main goal of the paper is to develop a DNN-based solution for iterative decoding systems, which require soft-input soft-output component decoders that typically utilize high-rate codes. Additional decoding performance results are provided in the Appendix B.

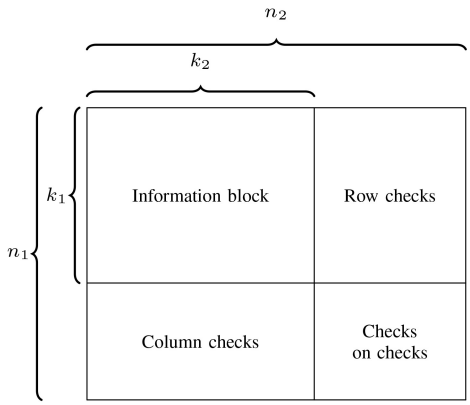


FIGURE 7. TPC structure.

IV. NN ITERATIVE SOFT-OUTPUT DECODING

The soft-output is required for the decoder to operate effectively in the iterative schemes. The TPC is selected to showcase the capabilities of the developed soft-output DNN-based decoder.

A. TURBO PRODUCT CODE

An illustration of the TPC structure can be seen in Fig. 7. The TPC is constructed using two component codes. Each of these component codes are in systematic form and have the following respective parameters: $[n_1, k_1]$ and $[n_2, k_2]$. The encoding process is comprised of two distinct phases. Initially, the $k_1 \times k_2$ information submatrix is encoded by the "column code", which generates a "column checks" submatrix. The next step involves the encoding of the combined information and column checks submatrices through the utilization of the "row code", leading to the generation of the "row checks" and "checks-on-checks" submatrices. The resulting code rate for the TPC is represented by the equation $R = (k_1 k_2)/(n_1 n_2)$ [29].

The process of iterative neural TPC decoding is illustrated in detail in Algorithm 2. In this context, N represents the number of decoding half-iterations, with $\mathcal{D}_c[\boldsymbol{\Theta}_i](\cdot)$ and $\mathcal{D}_r[\boldsymbol{\Theta}_i](\cdot)$ referring to the column and row decoding functions (Algorithm 1), parameterized by the trainable weights $\boldsymbol{\Theta}_i$. The matrices $\mathbf{L}_c \in \mathbb{R}^{n_2 \times n_1}$ and $\mathbf{L}_r \in \mathbb{R}^{n_1 \times n_2}$ represent the column and row extrinsic information, respectively. Additionally, $\varphi_i \in [0, 1]$ denotes the extrinsic scale factor, and $\mu_i \in [0, 1]$ signifies the decoder output scale factor during the i-th half-iteration. Term η is introduced to normalize the decoder input matrix $\mathbf{A}$. Normalization is applied to speed up the convergence during the DNN iterative decoding training, by fixing the expectation for all half-iteration component decoders.

B. NN TPC TRAINING SETUP

Existing iterative neural decoders are based on the autoencoder architecture and an unrolling strategy is utilized for training [24], [36]. Unrolling implies the initialization of separate sets of trainable parameters for each half-iteration

Algorithm 2 Neural TPC Decoding

Input: N – number of TPC half-iterations
$\mathbf{\Gamma} \in \mathbb{R}^{n_1 \times n_2}$ – channel output LLRs
$[\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_N]$ – set of component decoders weights
$[\varphi_1, \ldots, \varphi_N]$ – extrinsic scaling coefficients
$[\mu_1, \ldots, \mu_N]$ – decoder output scaling coefficients
Output: $\widehat{\mathbf{\Gamma}} \in \mathbb{R}^{n_1 \times n_2}$ – soft transmitted message estimation

$\mathbf{L}_c, \mathbf{L}_r \leftarrow 0$
$\widehat{\mathbf{\Gamma}} \leftarrow \mathbf{\Gamma}$
for $i = 1$ to N **do**
  if i is odd **then**
    Column decoding:
    $\mathbf{A} \leftarrow \widehat{\mathbf{\Gamma}}^T - \mathbf{L}_c$
    $\eta \leftarrow \frac{1}{n_1 n_2} \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} |\mathbf{A}_{i,j}|$
    $\mathbf{L}_0 \leftarrow \mathcal{D}_c[\mathbf{\Theta}_i](\mathbf{A}/\eta)$
    $\mathbf{L}_c \leftarrow \varphi_i(\mu_i \eta \mathbf{L}_0 - \mathbf{A})$
    $\widehat{\mathbf{\Gamma}} \leftarrow \mathbf{A}^T + \mathbf{L}_c^T$
  end if
  if i is even **then**
    Row decoding:
    $\mathbf{A} \leftarrow \widehat{\mathbf{\Gamma}} - \mathbf{L}_r$
    $\eta \leftarrow \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} |\mathbf{A}_{i,j}|$
    $\mathbf{L}_0 \leftarrow \mathcal{D}_r[\mathbf{\Theta}_i](\mathbf{A}/\eta)$
    $\mathbf{L}_r \leftarrow \varphi_i(\mu_i \eta \mathbf{L}_0 - \mathbf{A})$
    $\widehat{\mathbf{\Gamma}} \leftarrow \mathbf{A} + \mathbf{L}_r$
  end if
end for
return $\widehat{\mathbf{\Gamma}}$

Algorithm 3 DNN-Based TPC Decoder Sequential Training (Generalized for a Single Component Code)

Input: N – number of decoding half-iterations
N_{ep} – number of training epochs
$\mathbf{\Theta}_1$ – pretrained decoder weights (Section III-D)
$\mathcal{L}_{BCE}$ – BCE loss function (eq. (8))
Output: $[\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_N]$ – set of optimized component decoders parameters

$\varphi_1 \leftarrow 0.7$
$\mu_1 \leftarrow 1.0$
for $i = 2$ to N **do**
  Initialize current half-iteration decoder weights
  $\varphi_i \leftarrow \varphi_{i-1}$
  $\mu_i \leftarrow \mu_{i-1}$
  $\mathbf{\Theta}_i \leftarrow \mathbf{\Theta}_{i-1}$
  Disable $\mathbf{\Theta}_{i-1}$ gradient calculation
  for $j = 1$ to N_{ep} **do**
    Generate batch of codewords $\mathbf{C}$ and LLRs $\mathbf{\Gamma}$
    Perform decoding (Algorithm 2)
    $\widehat{\mathbf{\Gamma}} \leftarrow \texttt{decode}\left(i, \mathbf{\Gamma}, [\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_i], [\varphi_1, \ldots, \varphi_i], [\mu_1, \ldots, \mu_i]\right)$
    Calculate loss $\mathcal{L}_{BCE}(\widehat{\mathbf{\Gamma}}, \mathbf{C})$ and update weights $\mathbf{\Theta}_i, [\varphi_1, \ldots, \varphi_i], [\mu_1, \ldots, \mu_i]$
  end for
end for
return $[\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_N]$

with the following joint training. Such a training scheme has a complex and time-consuming backpropagation procedure since it requires optimization of all half-iteration decoders' weights simultaneously. Also, unrolled training can be unstable on relatively high learning rates, if the number of the component DNN-based decoder parameters becomes large. Recent works propose to train component code decoders separately and fit them into the iterative scheme with extrinsic information transfer (EXIT) charts [37], [38]. EXIT charts are an efficient design tool built on the concept of extrinsic information flow in iterative component-based systems [39]. Component-wise training solves the problem of its complexity, but to apply EXIT charts to fit the component decoders, the latter should satisfy an assumption of the Gaussian output distribution, which is not always achievable.

To train an iterative DNN-based decoder we propose to follow a two-stage procedure: (i) sequential training of component decoders; (ii) joint iterative decoder and extrinsic scaling coefficients fine-tuning.

The first training stage implies a sequential training of the component decoders. For the first half-iteration, we use a pretrained decoder described in Section III. The next half-iteration decoder is initialized with the weights of the previous half-iteration decoder and trained on its outputs. During the training of the last appended half-iteration decoder, all the preceding decoders' weights are frozen. As the BCE (8) loss plateau is achieved, a new half-iteration decoder is appended to the scheme. We note that the input of the first half-iteration component decoder has a Gaussian distribution, but it is not guaranteed for all subsequent iterations. Sequential training allows to precisely tune all component decoders to the shape of their input distributions, as well as to reduce the training time and increase its stability, since only one component decoder's set of weights is updated simultaneously. It is important to point out that during the TPC training extrinsic and decoder output scaling coefficients φ, μ remain unfrozen and optimized jointly with the decoder. The described training procedure for a single component code is summarized in the Algorithm 3.

Once all the component code decoders are trained, we proceed to the second stage. The decoder is unrolled as in [24] and fine-tuned in the TPC scheme jointly with the extrinsic and decoder output scaling coefficients φ, μ. Fine-tuning is performed with a low learning rate (10^{-6}) for a small number of epochs.

The loss function used during the fine-tuning stage of the NN-TPC is an exponentially weighted sum of the BCE loss (8) across all decoding iterations.

$$\boldsymbol{\beta} = [e^0, \ldots, e^{N-1}] \tag{13}$$

$$\mathcal{L}_{\text{NN-TPC}} = \frac{1}{N\|\boldsymbol{\beta}\|_1} \sum_{i=1}^{N} \beta_i \mathcal{L}_{BCE}(\widehat{\mathbf{\Gamma}}_i, \mathbf{C}). \tag{14}$$

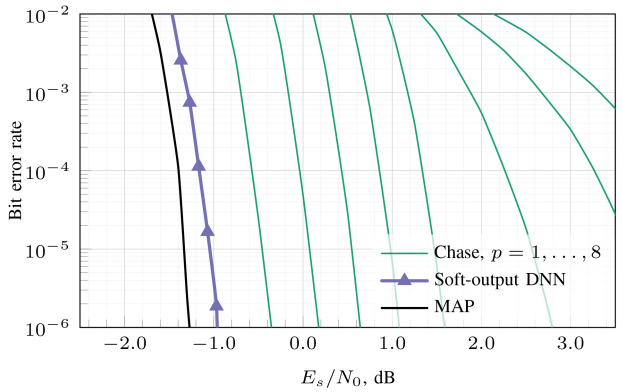


FIGURE 8. BER results for TPC decoding scheme with [64, 45] eBCH as component code, 4 full iterations.

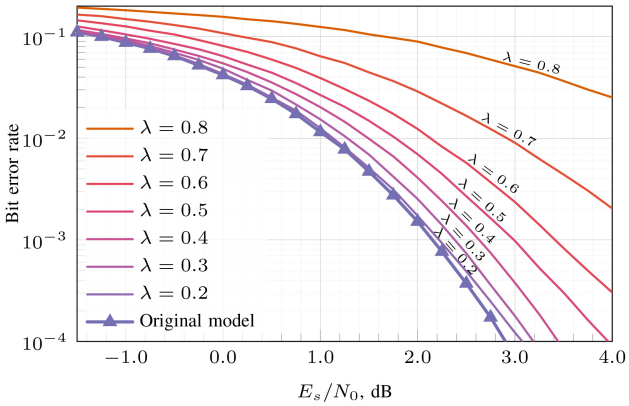


FIGURE 9. BER performance evaluation of the pruned [64, 45] eBCH decoding model.

Here, $\widehat{\mathbf{\Gamma}}_i$ represents the decoded column/row message LLRs at half-iteration i, and $\mathbf{C}$ refers to the transmitted binary TPC message.

C. SIMULATION RESULTS

To assess the developed soft-output decoder we utilize a TPC with a [64, 45] eBCH component code. The performance of the proposed decoder is compared to the Chase decoder with the soft-output calculation [28] and MAP decoding[3] over four full iterations ($N = 8$). BER results are depicted in Fig. 8. We observe that the DNN-based decoder outperforms the Chase-Pyndiah algorithm for $p \leq 8$ with a large gap and attains component MAP decoder performance by $\approx 0.2 - 0.25$ dB. To the best of our awareness, this is the first neural decoder demonstrating competitive performance in the iterative scheme for code and message lengths of $[n = 4096, k = 2025]$.

V. COMPLEXITY AND MEMORY REQUIREMENTS REDUCTION

The utilization of NN-based processing techniques in communication system's hardware is frequently restricted by the reasons of computational complexity and memory requirements. Deep learning models are often over-parameterized, leading to redundancy and wastage of both computation and memory resources [26]. When considering the complexity reduction of the neural decoders, the main and the most straightforward approach is to decrease the number of trainable parameters. In this paper, we follow two methods: unstructured pruning and low rank approximation. In the following subsections sparsity λ is used as the compression metric, defined as the ratio of the number of reduced weights $\mathcal{R}$ in the model to the total number of trainable parameters $\mathcal{T}$.

$$\lambda = \frac{\mathcal{R}}{\mathcal{T}}. \tag{15}$$

A complete derivation of the computational complexity of the decoding model described in this paper is provided in the Appendix C.

[3]The MAP decoder is used for decoding component codes. The extrinsics are scaled by a coefficient of $\varphi = 0.7$.

A. PRUNING

Pruning is a process of trainable weights sparsification. Pruning reduces a portion of the values in the model's weight matrices to zero, allowing the weights to be represented more efficiently. In a sparse representation, only the non-zero values are stored, along with a corresponding mask. Structured and unstructured pruning approaches can be distinguished.

Structured pruning implies the removal of neural network structural parts (filters, channels, layers, or groups of weights). Structured pruning allows to greatly improve the computational performance, by reducing part of the required computations. Usually, structured pruning is applied for the CNNs to reduce the number of filters [40].

Unstructured pruning is a more general approach to trainable weights sparsification. Usually, unstructured pruning implies the suppression of the predefined amount of weights with the least L_1 or L_2 norms. The best results described in the latest papers are achieved with the iterative approach: on each training epoch non-masked model weights are adjusted and the indices of the pruned neurons are reselected [41]. Unstructured pruning does not guarantee a reduction in the computational complexity of matrix multiplication for the model's parameters, but it can efficiently reduce memory requirements with minimal impact on performance.

To demonstrate pruning impact on the DNN-based performance we utilize simple noniterative unstructured L_1 norm pruning with the predefined sparsity for the trainable matrices inside the GRU cells and FC layer, without following model tuning. Fig. 9 demonstrates [64, 45] eBCH decoding performance in respect to the model sparsity. We can see that 30% of the model's parameters can be pruned without large performance loss.

B. LOW RANK COMPRESSION

In machine learning, tensor methods can be used to reduce the computational complexity and memory requirements of the DNN models that have already been trained. This is achieved by decomposing the weight tensors of the model into smaller, more manageable components that can be stored

and processed more efficiently. Tensor methods can also be used to improve the accuracy and generalization performance of deep learning models by incorporating additional constraints or regularization terms into the optimization process.

Tensor decomposition is a powerful technique for compressing models, especially recurrent neural networks. It allows one to reduce the size of the input-to-hidden layers in RNN models by over 1,000 times [42]. There are several tensor decomposition methods, such as Tucker decomposition [43], canonical polyadic decomposition (CPD) [44], and tensor train (TT) [45], which can be applied to multidimensional tensors. However, the most straightforward method applicable for two-dimensional matrices is the truncated singular value decomposition (SVD).

For a NN weight matrix $\mathbf{W} \in \mathbb{R}^{M\times B}$ SVD decomposition is defined as

$$\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{W}, \tag{16}$$

where $\mathbf{U} \in \mathbb{R}^{M\times K}, \mathbf{V}^T \in \mathbb{R}^{K\times B}$ are unitary matrices with $K = \min(M, B)$ and $\Sigma \in \mathbb{R}^{K\times K}$ diagonal matrix with singular values σ: $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_K \end{bmatrix}$, $\sigma_1 \geq \ldots, \geq \sigma_K$.

Dense matrices typically require MB elements to be stored. A rank-r approximation can reduce this number to $r(M + B)$. Suppose each of the trained NN weight matrices involved in matrix-vector multiplication represents a function $f(x), \quad x \in \mathbb{R}$ and we seek a low-dimensional parametrization of this function. Specifically, we aim to identify the directions in which a $f(x)$ changes a lot on average and for the directions in which $f(x)$ is almost constant. SVD solves the above-mentioned problem. From the singular values σ distribution one can estimate the matrix rank. Truncating the columns of $\mathbf{U} \in \mathbb{R}^{M\times K} \rightarrow \mathbf{U}_c \in \mathbb{R}^{M\times r}, \quad \mathbf{V}^T \in \mathbb{R}^{K\times B} \rightarrow \mathbf{V}_c^T \in \mathbb{R}^{r\times B}$ and vector of $\text{diag}(\Sigma) \in \mathbb{R}^K \rightarrow \text{diag}(\Sigma_c) \in \mathbb{R}^r$ to the desired condition number we can approximate matrix $\mathbf{W}$ to the defined rank r matrix $\mathbf{W}_c = \mathbf{U}_c\Sigma_c\mathbf{V}_c^T$. Since the diagonal matrix Σ_c can be multiplied by either of the unitary matrices, $\mathbf{U}_c$ or $\mathbf{V}_c^T$, before being stored in memory, we are required to store $r(M+B)$ real-valued elements. This allows us to redefine the sparsity as $\lambda = 1 - \frac{r(M+B)}{MB}$. The complexity of the matrix multiplication is reduced by the same factor λ.

Modern methods of decomposition-based compression imply compression-aware training: decomposition is performed in an online fashion during training, jointly with the weights update. This approach generally yields better compression and performance results. In this paper, we will follow the simple "compression after training" approach to demonstrate the compressed DNN-based decoder's performance bounds. The results of the truncated SVD compression are depicted in Fig. 10. We can compare them with pruning compression. One can observe that low SVD compression of the decoding model leads to worse performance than pruning, which degrades comparatively slowly with an increase in the compression ratio. However,

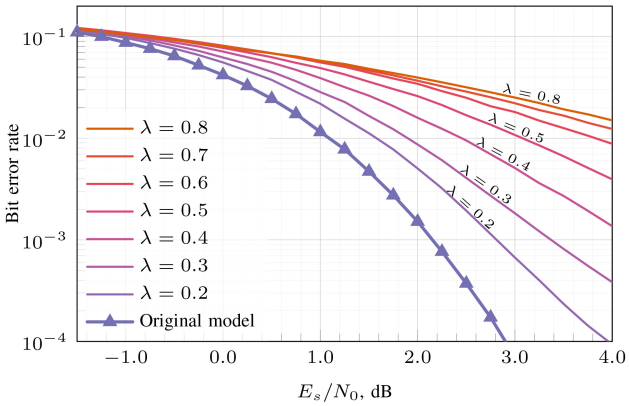


FIGURE 10. BER performance evaluation of [64, 45] eBCH decoding model compressed by low-rank SVD approximation.

SVD-based compression performance converges at high sparsity ratios, which describes its wide application for heavy NN compression. We conclude that for the model utilized in the described example, pruning solves the compression task better when decoding performance is more important. Truncated SVD may be applied for large compression ratios, but the decoding performance degrades faster.

VI. CONCLUSION

This paper introduces a modified syndrome-based DNN-decoding approach that enhances performance by incorporating regularization and training to replicate MAP decoder soft outputs. The approach was evaluated on eBCH and CA-Polar codes with various rates demonstrating its potential for iterative decoding schemes. The model was specifically tailored for turbo product code decoding using a sequential training method. The DNN-based iterative decoder outperformed the Chase-Pyndiah algorithm with $p \leq 8$, achieving MAP decoder performance within 0.2–0.25 dB. To the best of our knowledge, this is the first neural decoder demonstrating competitive performance within an iterative scheme for code and message lengths of $[n = 4096, k = 2025]$. The study also addressed computational challenges by employing model compression techniques, such as pruning and matrix decomposition, to balance resource constraints with decoding accuracy.

APPENDIX A
STACKED-GRU NOISE ESTIMATOR ARCHITECTURE

The Stacked-GRU is an RNN architecture comprised of GRU cells [46], which include trainable "update" and "reset" gates. The behavior of each GRU cell is defined by the following equations (see Fig. 11 for more details).

$$\mathbf{g}_t = \sigma(\mathbf{W}_g\mathbf{d}_t + \mathbf{U}_g\mathbf{q}_{t-1} + \mathbf{b}_g), \tag{17}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{d}_t + \mathbf{U}_r\mathbf{q}_{t-1} + \mathbf{b}_r), \tag{18}$$

$$\widehat{\mathbf{q}}_t = \tanh\left(\mathbf{W}_h\mathbf{d}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{q}_{t-1}) + \mathbf{b}_h\right), \tag{19}$$

$$\mathbf{q}_t = \mathbf{g}_t \odot \widehat{\mathbf{q}}_t + (\mathbf{1} - \mathbf{g}_t) \odot \mathbf{q}_{t-1}. \tag{20}$$

Here, $\mathbf{d}_t$ represents the input vector, $\mathbf{q}_t$ denotes the output vector, $\widehat{\mathbf{q}}_t$ stands for the candidate output vector, $\mathbf{g}_t$ is the

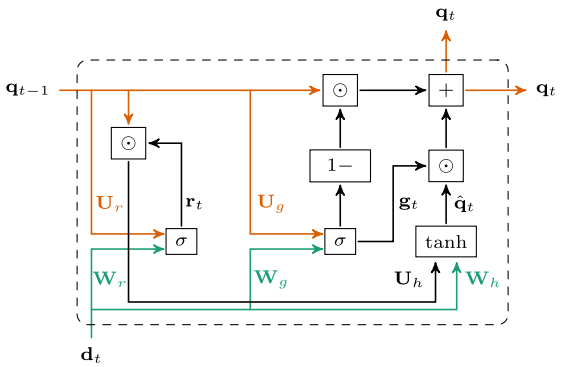


FIGURE 11. Gated Recurrent Unit cell.

update gate vector, and $\mathbf{r}_t$ is the reset gate vector. The matrices $\mathbf{W}$ and $\mathbf{U}$ are trainable multiplicative weights, and $\mathbf{b}$ represents trainable bias vectors. The function $\sigma(\cdot)$ indicates the sigmoid activation and $\tanh(\cdot)$ refers to the hyperbolic tangent activation. Such architecture is commonly employed for natural language processing tasks.

To construct a Stacked-GRU architecture, cells are organized in two dimensions. In the vertical dimension, the output $\mathbf{q}_t$ from the previous cell is fed into the feature input $\mathbf{d}_t$ of the next cell. The total number of layers in the Stacked-GRU network is denoted by L. The horizontal stacking dimension determines the network's recurrent structure, where the output from previous cells is transferred as the hidden state $\mathbf{q}_{t-1}$ to the subsequent cells. The network's initial hidden state $\mathbf{q}_0$ is initialized by zeros. The total number of *time steps* is denoted by T. It is worth noting that the number of trainable parameters of the Stacked-GRU does not depend on T, since the same set of GRU cells is utilized in the recurrent manner for all of the *time steps*.

For a single input vector, the Stacked-GRU neural network generates T vectors from the output of the final layer. We represent this stacked output vector as $\mathbf{Q}^{(L)} = [\mathbf{q}_1^{(L)}, \ldots, \mathbf{q}_t^{(L)}, \ldots, \mathbf{q}_T^{(L)}] \in \mathbb{R}^{n_h T}$, where the superscript L indicates the last layer, the subscript t refers to the GRU time step, and n_h denotes the hidden size of the cell. To reduce the dimension of the Stacked-GRU output vector from $n_h T$ to the length of the LLR vector n, the output $\mathbf{Q}^{(L)}$ is fed into a single fully connected layer. The overall structure of the noise estimation model is illustrated in Fig. 12.

APPENDIX B
ADDITIONAL DECODING PERFORMANCE RESULTS

To validate the robustness of the proposed DNN-based decoding method and to illustrate the limitations of belief propagation-based algorithms application for codes utilized in our studies, we present additional simulation results for eBCH codes of lengths 32, 64, and 128 in Fig. 13.

APPENDIX C
DECODING MODEL COMPLEXITY ANALYSIS

The total computational complexity of the proposed DNN-based decoder includes the complexity of the Stacked-GRU model and the linear layer that reduces the dimensionality of the Stacked-GRU output.

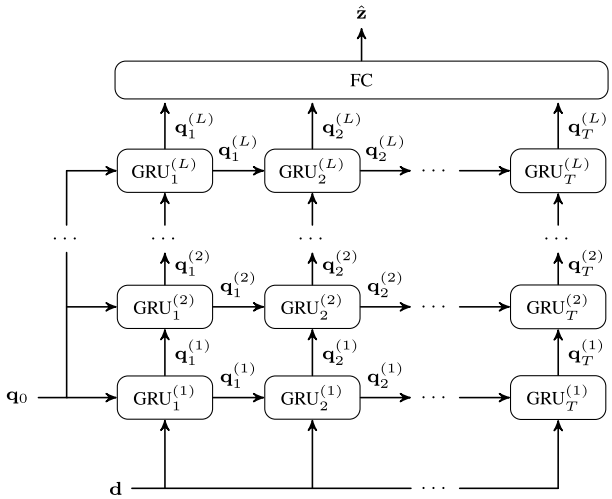


FIGURE 12. Stacked-GRU model architecture.

The number of real multiplications of the single GRU-cell is

$$C_{GRU} = \mathcal{O}(n_h(3n_i + 3n_h + 3)), \tag{21}$$

where n_h – cell hidden size, n_i – length of input vector [47].

The input vector size of the GRU cell differs for the first layer and all subsequent layers of the Stacked-GRU. For the first layer, the input size is $n_i^{(1)} = 2n - k$ where n is the code length and k is the code dimension. For the following layers, the input size equals the hidden size n_h of the previous cell. Additionally, we must consider that the model's hidden state is initialized by zeros, resulting in reduced complexity for the GRU cells at time step $t = 1$ by $3n_h^2$.

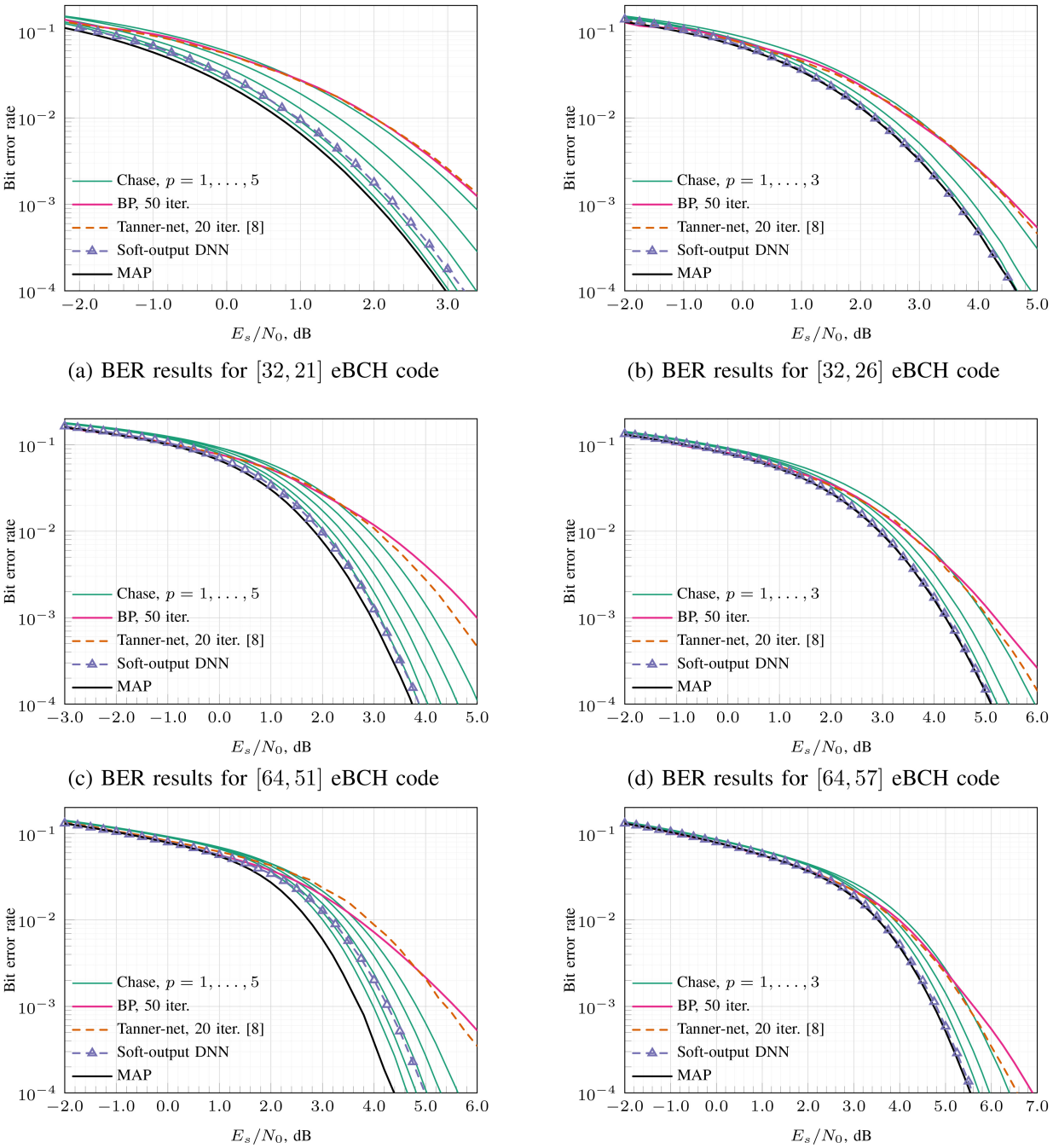
The resulting complexity of the Stacked-GRU model is

$$\begin{aligned} C_{S.GRU} = \mathcal{O}\Big(& n_h(3n_i + 3) \\ & + n_h(T-1)(3n_h + 3n_i + 3) \\ & + n_h(L-1)(3n_h + 3) \\ & + n_h(T-1)(L-1)(3n_h + 3n_h + 3)\Big) \\ = n_h\Big(& (3n_i + 3) + (T-1)(3n_h + 3n_i + 3) \\ & + (L-1)(3n_h + 3) + (T-1)(L-1)(6n_h + 3)\Big), \end{aligned} \tag{22}$$

where T is the number of time steps and L is the number of layers.

The complexity of the linear layer is determined by the length of the input vector, denoted as n_{FC_i}, and the number of neurons, which, in our case, is equal to the code length n. In this application, the input to the linear layer is the flattened vector resulting from the concatenation of the last-layer outputs of a Stacked-GRU, meaning that $n_{FC_i} = T n_h$.

$$C_{FC} = \mathcal{O}(n_{FC_i} n) = \mathcal{O}(T n_h n). \tag{23}$$



(a) BER results for $[32, 21]$ eBCH code

(b) BER results for $[32, 26]$ eBCH code

(c) BER results for $[64, 51]$ eBCH code

(d) BER results for $[64, 57]$ eBCH code

(e) BER results for $[128, 113]$ eBCH code

(f) BER results for $[128, 120]$ eBCH code

FIGURE 13. **BER performance of eBCH code decoding.**

REFERENCES

[1] D. Artemasov, K. Andreev, P. Rybin, and A. Frolov, "Soft-output deep neural network-based decoding," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, 2023, pp. 1692–1697.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[3] X.-A. Wang and S. Wicker, "An artificial neural net Viterbi decoder," *IEEE Trans. Commun.*, vol. 44, no. 2, pp. 165–171, Feb. 1996.

[4] L. G. Tallini and P. Cull, "Neural nets for decoding error-correcting codes," in *Proc. IEEE Tech. Appl. Conf. Workshops. Northcon/95, Conf. Rec.*, Oct. 1995, p. 89.

[5] T. Gruber, S. Cammerer, J. Hoydis, and S. t. Brink, "On deep learning-based channel decoding," in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2017, pp. 1–6.

[6] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 119–131, Feb. 2018.

[7] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 1361–1365.

[8] K. Andreev, A. Frolov, G. Svistunov, K. Wu, and J. Liang, "Deep neural network based decoding of short 5G LDPC codes," in *Proc. XVII Int. Symp. 'Problems Redundancy Inf. Control Syst.' (REDUNDANCY)*, 2021, pp. 155–160.

[9] J. Dai et al., "Learning to decode protograph LDPC codes," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 1983–1999, Jul. 2021.

[10] M. Lian, F. Carpi, C. Häger, and H. D. Pfister, "Learned belief-propagation decoding with simple scaling and SNR adaptation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2019, pp. 161–165.

[11] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, "RNN decoding of linear block codes," 2017, *arXiv:1702.07560*.

[12] B. Vasić, X. Xiao, and S. Lin, "Learning to decode LDPC codes with finite-alphabet message passing," in *Proc. Inf. Theory Appl. Workshop (ITA)*, 2018, pp. 1–9.

[13] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: MIT Press, 1963.

[14] V. Kuzurman, D. Artemasov, K. Andreev, and A. Frolov, "On joint neural min-sum decoding and quantization optimization," in *Proc. IEEE Int. Multi-Conf. Eng., Comput. Inf. Sci. (SIBIRCON)*, 2024, pp. 58–64.

[15] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," in *Advances in Neural Information Processing Systems*, vol. 32. Red Hook, NY, USA: Curran Assoc., Inc., 2019.

[16] E. Nachmani and L. Wolf, "A gated hypernet decoder for polar codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2020, pp. 5210–5214.

[17] S. Cammerer, J. Hoydis, F. A. Aoudia, and A. Keller, "Graph neural networks for channel decoding," 2022, *arXiv:2207.14742*.

[18] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes-a syndrome-based approach," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1595–1599.

[19] Y. Choukroun and L. Wolf, "Error correction code transformer," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Red Hook, NY, USA: Curran Assoc., Inc., 2022, pp. 38695–38705. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/fcd3909db30887ce1da519c4468db668-Paper-Conference.pdf

[20] Y. Choukroun and L. Wolf, "Denoising diffusion error correction codes," 2022, *arXiv:2209.13533*.

[21] T. Matsumine and H. Ochiai, "Recent advances in deep learning for channel coding: A survey," 2024, *arXiv:2406.19664*.

[22] C. Studer, "Iterative MIMO decoding: Algorithms and VLSI implementation aspects," Ph.D. dissertation, ETH Zürich, Zürich, Switzerland, Jun. 2009. [Online]. Available: http://www.nari.ee.ethz.ch/pubs/p/studerdiss09

[23] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[24] M. V. Jamali, H. Saber, H. Hatami, and J. H. Bae, "ProductAE: Toward training larger channel codes based on neural product codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2022, pp. 3898–3903.

[25] J. Clausius, M. Geiselhart, and S. ten Brink, "Component training of turbo autoencoders," 2023, *arXiv:2305.09216*.

[26] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26. Red Hook, NY, USA: Curran Assoc., Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2013/file/7fec306d1e665bc9c748b5d2b99a6e97-Paper.pdf

[27] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.

[28] R. Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 8, pp. 1003–1010, Aug. 1998.

[29] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, U.K.: Cambridge Univ. Press, 2009. [Online]. Available: https://books.google.ru/books?id=n9BNngEACAAJ

[30] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[31] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland Math. Libr., 1977. [Online]. Available: https://doi.org/10.1016/s0924-6509

[32] L. Lugosch and W. J. Gross, "Learning from the syndrome," in *Proc. 52nd Asilomar Conf. Signals, Syst., Comput.*, 2018, pp. 594–598.

[33] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[34] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Hoboken, NJ, USA: Wiley-Intersci., 2006.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017, *arXiv:1412.6980*.

[36] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels," in *Proc. 33rd Conf. Neural Inf. Process. Syst.*, 2019, pp. 2754–2764.

[37] J. Clausius, S. Dörner, S. Cammerer, and S. ten Brink, "Serial vs. parallel turbo-autoencoders and accelerated training for learned channel codes," in *Proc. 11th Int. Symp. Topics Coding (ISTC)*, 2021, pp. 1–5.

[38] J. Clausius, M. Geiselhart, and S. T. Brink, "Component training of turbo autoencoders," in *Proc. 12th Int. Symp. Topics Coding (ISTC)*, 2023, pp. 1–5.

[39] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.

[40] C. Lemaire, A. Achkar, and P.-M. Jodoin, "Structured pruning of neural networks with budget-aware regularization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9100–9108.

[41] C. M. J. Tan and M. Motani, "DropNet: Reducing neural network complexity via iterative pruning," in *Proc. 37th Int. Conf. Mach. Learn.*, Jul. 2020, pp. 9356–9366. [Online]. Available: https://proceedings.mlr.press/v119/tan20a.html

[42] X. Liu and K. K. Parhi, "Tensor decomposition for model reduction in neural networks: A review [feature]," *IEEE Circuits Syst. Mag.*, vol. 23, no. 2, pp. 8–28, 2nd Quart., 2023.

[43] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966. [Online]. Available: https://doi.org/10.1007/BF02289464

[44] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *J. Math. Phys.*, vol. 6, nos. 1–4, pp. 164–189, 1927. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sapm192761164

[45] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011. [Online]. Available: https://doi.org/10.1137/090752286

[46] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014, *arXiv:1409.1259*.

[47] P. J. Freire, S. Srivallapanondh, A. Napoli, J. E. Prilepsky, and S. K. Turitsyn, "Computational complexity evaluation of neural network applications in signal processing," 2022, *arXiv:2206.12191*.

DMITRY ARTEMASOV (Graduate Student Member, IEEE) received the B.Sc. degree in telecommunications from Ural Federal University in 2021, and the M.Sc. degree in computer science from the Skolkovo Institute of Science and Technology in 2023, where he is currently pursuing the Ph.D. degree. His research interests include MIMO systems, coding theory, information theory, deep learning, IoT, and digital signal processing. He is a recipient of the Russian President and the Russian Government Scholarships.

KIRILL ANDREEV (Member, IEEE) received the M.Sc. degree in applied mathematics from the Moscow Institute of Physics and Technology in 2010, and the Ph.D. degree in technical sciences from the Institute of Control Sciences of the Russian Academy of Sciences in 2016. He is currently an Assistant Professor with the Skolkovo Institute of Science and Technology, Moscow, Russia. His research interests include multiple access protocols, information and coding theory, and applications of machine learning in communication systems.

PAVEL RYBIN (Member, IEEE) received the M.Sc. degree in computer science from Bauman Moscow State Technical University in 2010, and the Ph.D. degree in mathematics from the Institute for Information Transmission Problems, Russian Academy of Sciences in 2012. He is currently a Leading Research Scientist with the Skolkovo Institute of Science and Technology, Moscow, Russia. His research interests include error-correcting codes, decoding algorithms, complexity theory, and communication systems. He is a recipient of the Russian President Scholarship and laureate of the contest of young scientists for participation in conferences and seasonal schools in the field of computer science of the Dynasty fund.

ALEXEY FROLOV (Member, IEEE) received the M.Sc. degree in computer science from Bauman Moscow State Technical University in 2010, the Ph.D. degree in mathematics from the Institute for Information Transmission Problems, Russian Academy of Sciences, in 2012, and the D.Sc. degree in mathematics from Moscow Institute of Physics and Technology in 2021. He is currently a Professor with the Skolkovo Institute of Science and Technology, Moscow, Russia. His research interests include information theory and its applications in telecommunications, storage systems, and other areas. He was a recipient of the IEEE GLOBECOM Communication Theory Symposium Best Paper Award in 2020, the Russian Government Award in Science and Technology for Young Scientists in 2016, and the Moscow Government Award for Young Scientists in 2013.