AGENCE NATIONALE DE LA RECHERCHE

ANR

AI4CODE

AI-aided FEC code design & decoding

ANR, Appel à Projets Générique (AAPG 2021)

AI4CODE Project (ANR-21-CE25-0006)

Deliverable D3.2

Improved Learning-Based Decoders

Final Report

Editor:	Valentin Savin (CEA)
Deliverable nature:	Public
Due date:	October 31, 2025
Delivery date:	October 22, 2025
Version:	1.0
Total number of pages:	104 pages
Keywords:	LDPC codes, syndrome-based neural decoding, belief-propagation, multiple-round belief-propagation, ordered statistics decoding.

Abstract

This deliverable reports on the improved learning-based decoders obtained from the design space exploration analysis in D3.1, the performance and complexity assessment of these decoders, and the analysis of the learning outputs. It encompasses the work carried out in Task 3.2 and Task 3.3, during the full course of the project.

List of Authors

Partner	Author
LAB-STICC/IMTA	Raphaël Le Bidan (raphael.lebidan@imt-atlantique.fr)
	Ahmad Ismail (ahmad.ismail@imt-atlantique.fr)
	Elsa Dupraz (elsa.dupraz@imt-atlantique.fr)
	Charbel Abdel Nour (charbel.abdelnour@imt-atlantique.fr)
CEA-LETI	Valentin Savin (valetin.savin@cea.fr)
	Valérien Mannoni (valerien.mannoni@cea.fr)
	Joachim Rosseel (joachim.rosseel@cea.fr)
IRIT/INP-ENSEEIHT	Charly Poulliat (charly.poulliat@enseeiht.fr)
ETIS/CYU	Inbar Fijalkow (inbar.fijalkow@ensea.fr)

Contents

Introduction

Context of this work

The aim of the AI4CODE project is to explore and assess how machine learning (ML) techniques can contribute to improvements in coding theory, techniques, and practice. The focus is placed on forward error correction (FEC), and the project is built around the following four inter-related objectives, identified and detailed in the scientific document of the project:

Objective #1: Explore how ML can contribute to improving the state of the art (SoA) in FEC decoding.

Objective #2: Investigate how ML techniques can improve current knowledge and practice in FEC code design.

Objective #3: Learn from the machine.

Objective #4: Develop a general expertise and critical thinking on ML algorithms and their applications to coding theory and practice.

Work Package 3 (WP3) focuses on ML-augmented FEC decoding techniques, thus addressing the first of the above objectives. Leveraging ML tools and algorithms to improve the SoA in FEC decoding has been gaining increasing attention in the last few years. A first major issue that has been identified is to improve the performance of message-passing (MP) decoding at short to moderate block length, with the ultimate purpose of closely approaching maximum-likelihood decoding (MLD) at reasonable cost. ML is also regarded as a promising approach to reduce the complexity of certain decoding algorithms, for example to alleviate the burden of message computation in non-binary LDPC decoders, or to accelerate successive-cancelation list-decoding of Polar codes by better predicting the optimal path and pruning unlikely candidates earlier. Finally, learning may also help design decoders that adapt automatically, online, to unknown or rapidly-varying channel parameters. Encouraging results have been reported in the literature, although a significant gap to MLD performance, at practical cost, still remains.

The Gantt chart of WP3 is represented in Fig. 1. This deliverable reports on the activities carried out within the **Task 3.2** and **Task 3.3**, during the project. These activities focused on improved learning-based decoders obtained from the design space exploration analysis in D3.1, the performance and complexity assessment of these decoders, and the analysis of the learning outputs.

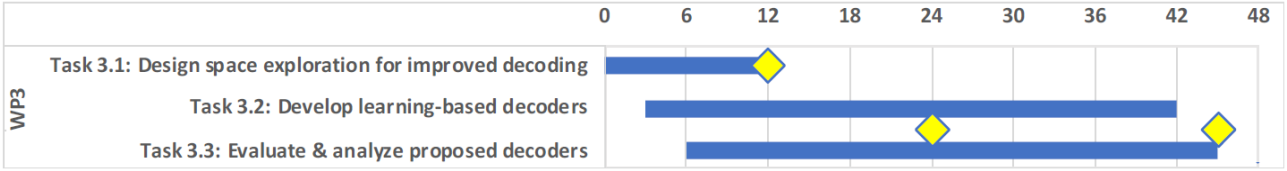


Figure 1: Gantt diagram of WP3

Organization of the deliverable

The field of learning-based channel decoding, or, more compactly, neural decoding, is shaped by two main paradigms: model-free and model-based approaches. Model-free decoders treat decoding as a pure learning task, allowing the use of diverse neural network architectures and enabling code-agnostic solutions. However, they often struggle with scalability and generalization. In contrast, model-based

decoders embed structural knowledge, such as graphical code constraints or traditional decoding rules, into the learning process, which can improve interpretability and guide learning, but at the cost of reduced flexibility in model design. No clear winner between the two approaches has emerged yet when it comes to closely approach MLD. The deliverable is organized in four sections that revolve around those two approaches.

Section 1 focuses on model-free neural decoding, and addresses the long-standing problem of devising computationally-efficient soft-decision decoding algorithms for arbitrary linear block codes. Here, the goal is to approximate MLD as closely as possible without prior inductive bias on the code. A particular emphasis is placed on syndrome-based neural decoding (SBND). We first describe the principle, implementation, and limitations of existing SBND models. We then propose heuristics for constructing carefully designed fixed datasets to improve the training of SBND models and examines complementary methods to further enhance their performance at inference time.

The next two sections shift the focus to model-based decoders. Both are devoted to the challenging problem of devising practical decoding algorithms for short LDPC codes that could match, or at least closely approach, the performance of MLD. Progress on this issue could bring more than 1 dB gain for many of the FEC codes found in the short-packet communication systems at the heart of the Internet-of-Things, making this problem of not only of theoretical but also of great practical value.

Section 2 investigates an LDPC decoding technique in which some perturbation is applied to the input of the Belief-Propagation (BP) decoder whenever it fails, in an attempt to correct some of the channel errors. BP decoding is then restarted for another round. The overall process is repeated several times until a codeword is found or a maximum number of perturbations has been tested. We review various decoders of this kind from the literature, and note that they can practically approach the performance of MLD, but that this requires processing an unreasonably large number of perturbation patterns. We provide experimental evidence suggesting that, with a proper choice of input perturbations, it might be possible to approach MLD in a much more computationally-efficient manner. This lead us to consider and evaluate different ways of drawing, first, on machine learning, then, on deep learning, to eventually arrive at smarter learned decoders which need less perturbation patterns to reach the same performance than the best-known expert decoders.

Section 3 focuses on improving the decoding performance of short LDPC codes, by leveraging on and combining several techniques, such as neural belief-propagation, decoding diversity, and post-processing. We observe that decoding approaches based solely on neural BP lead to an increase in the decoding speed, rather than an intrinsic improvement of the error correction capability. To improve the error correction performance, we complement neural-BP or conventional BP decoding with a post-processing step, based on ordered statistics decoding (OSD). This leads us to consider new approaches to neural model based optimization, where the aim is not to deliver the best possible decoding performance, but merely an output that best suits the post-processing step. Overall, the proposed approach, combining neural BP and low-order OSD, allows approaching or covering a significant part of the gap to MLD.

Finally, Section 4 takes a detour with the initial focus of WP3 in exploring the use of deep learning techniques to multi-user detection in multi-antenna systems by means of message-passing algorithms based on variational inference. First, we unfold the iterations in the turbo-receiver to optimize certain scalar parameters that do not have an analytic expression. Second, we study the application of neural networks to the graphs induced by the communication models. The results obtained with both approaches demonstrate that introducing learning in various places of the turbo-receivers can result in complexity savings without compromising the performance.

1 Syndrome-Based Neural Decoding of Linear Codes

1.1 Motivation

Model-free neural decoders rely on neural network (NN) architectures that do not incorporate any explicit information about the underlying code structure. Instead, they learn to directly map the noisy observations received from the channel back to the original transmitted codewords. This approach treats decoding purely as a data-driven inference problem, leveraging the flexibility and expressive power of modern NNs. Since no algebraic properties or decoding rules of the code are encoded into the architecture or learning procedure, model-free decoders are considered code-agnostic and can, in principle, be applied to any block code. Ideally, the NN architectures employed should be trained on the full set of codewords. This leads to the *curse of dimensionality* due to the exponential growth of the size of the codebook with the code's dimension. Nevertheless, model-free decoders have demonstrated high potential in achieving near-optimal decoding performance for short block codes.

Syndrome-Based Neural Decoding (SBND) [1] is a prominent example of model-free decoding for general linear block codes. SBND aims to estimate the channel-induced error pattern, using both the syndrome and channel reliability information as inputs to the NN. A key advantage of this formulation is that it allows the generation of training data from noisy realizations of a single codeword, without loss of generality. Another strength of SBND lies in its architectural flexibility as the model can be instantiated using a variety of NN architectures. The literature spans a range of such architectures, from simple multilayer perceptrons (MLPs) to more advanced recurrent neural networks (RNNs), and more recently, transformer-based models like ECCT [2].

In this first contribution to WP3, we start by introducing the general soft-decision decoding problem. Then we describe the SBND framework and training procedure. We review the main deep neural network (DNN) architectures proposed in the literature, and analyze their decoding performance on selected codes, with a particular focus on how closely they can approach MLD. We show that while SBND has close to optimal BER performance, a noticeable performance gap remains in FER, even after large-scale training. This leads us to develop two contributions aimed at reducing the gap to MLD. **First**, we shift the focus from purely architectural improvements, which garnered the most attention in the literature on SBND so far, to the often-overlooked role of training data. We propose a set of data-centric heuristics designed to enhance the training performance and generalization capabilities of SBND models while reducing the overall data requirements. These heuristics are architecture-agnostic and draw inspiration from best practices in both deep learning and channel coding, enabling more efficient and effective model training. **Second**, we introduce inference-time enhancement techniques that can be applied to already trained SBND models to push their performance closer to MLD without incurring additional training costs. These methods again leverage complementary ideas from deep learning and coding theory and provide means of closing the FER gap to MLD.

Related publications. The above contributions have been published in:

[**ILDA25a**] A. Ismail, R. Le Bidan, E. Dupraz, and C. Abdel Nour, "Doing More With Less: Towards More Data-Efficient Syndrome-Based Neural Decoders" in Proc. of *IEEE International Conference on Machine Learning for Communications and Networking (ICMLCN)*, Barcelona, Spain, May 2025.

[**LIDA25b**] R. Le Bidan, A. Ismail, E. Dupraz, and C. Abdel Nour, "Apport de l'augmentation de données au décodage neuronal souple par syndrome des codes correcteurs d'erreurs" in Proc. of *30ème édition du colloque GRETSI*, Strasbourg, France, Aug. 2025.

We provide an extended summary of the above contributions in the next sections. For more details we refer to the above publications, as well as to the PhD manuscript of Ahmad Ismail.

1.2 Soft-decision decoding of linear block codes

This section discusses soft-decision decoding for linear block codes. It introduces the transmission channel model, presents the maximum likelihood decoding rule, and comment on its computational challenges.

1.2.1 Transmission System Model

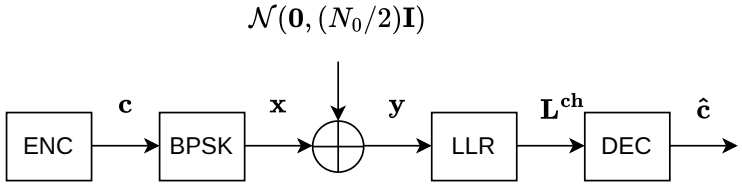


Figure 1.1: The BI-AWGN transmission system model.

A typical communication model is depicted in Fig. 1.1. Assume we have a binary linear code $\mathcal{C}(n,k)$ with an associated parity-check matrix $\mathbf{H}$. A binary information vector is encoded into a codeword $\mathbf{c} \in \mathcal{C}$, which is then mapped onto a BPSK constellation: the i-th bit c_i is modulated to $x_i \in \{+1,-1\}$ according to the rule $x_i = (-1)^{c_i}$, for $i = 1,\ldots,n$. The modulated signal $\mathbf{x} = (x_1,\ldots,x_n)$ is transmitted over an additive white Gaussian noise (AWGN) channel. The received vector $\mathbf{y} = (y_1,\ldots,y_n)$ is given by $\mathbf{y} = \mathbf{x} + \mathbf{n}$, where $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, (N_0/2)\mathbf{I})$ is a zero-mean Gaussian noise vector with variance $\sigma^2 = \frac{N_0}{2}$. At the receiver, the channel log-likelihood ratio (LLR) for the i-th bit is defined as

$$L_i^{\text{ch}} = \log \frac{P(y_i \mid c_i = 0)}{P(y_i \mid c_i = 1)} = \log \frac{P(y_i \mid x_i = +1)}{P(y_i \mid x_i = -1)}. \tag{1.1}$$

Under the AWGN channel model, the conditional probability distribution function is Gaussian:

$$P(y_i \mid x_i = \pm 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i \mp 1)^2}{2\sigma^2}\right). \tag{1.2}$$

Substituting into the LLR definition, we obtain:

$$L_i^{ch} = \log \frac{\exp\left(-\frac{(y_i-1)^2}{2\sigma^2}\right)}{\exp\left(-\frac{(y_i+1)^2}{2\sigma^2}\right)} = \frac{(y_i+1)^2 - (y_i-1)^2}{2\sigma^2} = \frac{2}{\sigma^2} y_i \tag{1.3}$$

The resulting LLR vector $\mathbf{L}^{\text{ch}} = (L_1^{\text{ch}},\ldots,L_n^{\text{ch}})$ serves as the soft input to the decoder, which aims to infer the transmitted codeword $\hat{\mathbf{c}}$.

1.2.2 The Optimal Decoder

The optimal design criterion for a decoder is to minimize the probability of decoding error. The error probability is defined as the probability that the decoded codeword $\hat{\mathbf{c}}$ differs from the transmitted one $\mathbf{c}$, i.e.,

$$P_e = P(\mathbf{c} \neq \hat{\mathbf{c}}) = P(\mathbf{x} \neq \hat{\mathbf{x}}). \tag{1.4}$$

For a given received vector $\mathbf{y}$, the conditional error probability is:

$$P(\hat{\mathbf{x}} \neq \mathbf{x} \mid \mathbf{y}) = 1 - P(\hat{\mathbf{x}} = \mathbf{x} \mid \mathbf{y}). \tag{1.5}$$

Thus, minimizing the probability of error is equivalent to selecting the codeword that maximizes the a posteriori probability:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c} \in \mathcal{C}} P(\mathbf{x} \mid \mathbf{y}). \tag{1.6}$$

Applying Bayes' theorem and since $P(\mathbf{y})$ does not depend on $\mathbf{c}$, the optimal decision rule becomes:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} P(\mathbf{y} \mid \mathbf{x})P(\mathbf{x}). \tag{1.7}$$

Assuming equiprobable codewords, $P(\mathbf{x})$ is constant and can be ignored. Therefore, the maximum a posteriori (MAP) rule in (1.6) can be replaced by the maximum likelihood (ML) rule:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} P(\mathbf{y} \mid \mathbf{x}). \tag{1.8}$$

Minimum-distance decoding Under the AWGN channel, using the conditional probability in (1.2), the ML decision rule in (1.8) can be written as:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{\|\mathbf{y}-\mathbf{x}\|^2}{2\sigma^2}\right). \tag{1.9}$$

This is equivalent to minimizing the squared Euclidean distance:

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}\in\mathcal{C}} \|\mathbf{y}-\mathbf{x}\|^2 = \arg\min_{\mathbf{c}\in\mathcal{C}} \|\mathbf{y}-(-1)^{\mathbf{c}}\|^2. \tag{1.10}$$

The MLD rule can expressed in alternative forms, such as the correlation-based formulation or the syndrome-based formulation.

Correlation decoding. The expression in (1.10) can be reformulated in terms of a correlation metric [3]. Specifically, the decoder can select the codeword that maximizes the inner product:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \langle(-1)^{\mathbf{c}}, \mathbf{y}\rangle = \arg\max_{\mathbf{c}\in\mathcal{C}} \sum_{i=1}^{n} (-1)^{c_i} y_i. \tag{1.11}$$

Since the log-likelihood ratio (LLR) for BPSK modulation under AWGN is proportional to the received value (i.e., $L_i^{ch} = \frac{2}{\sigma^2} y_i$), the MLD decision rule can equivalently be written in terms of LLRs:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \sum_{i=1}^{n} (-1)^{c_i} L_i^{\text{ch}}. \tag{1.12}$$

Syndrome decoding. Syndrome decoding reinterprets the decoding problem as coset decoding using the syndrome. Let $\mathbf{y} = \mathbf{x} + \mathbf{n}$ be received vector with its hard-decision:

$$z_i = \begin{cases} 0 & y_i \geq 0 \\ 1 & y_i < 0 \end{cases}. \tag{1.13}$$

Assume that $\mathbf{z} = \mathbf{c} + \mathbf{e}$. Then, $\mathbf{e}$ denotes the binary error pattern defined as:

$$e_i = \begin{cases} 1 & z_i \neq c_i \\ 0 & z_i = c_i \end{cases}. \tag{1.14}$$

The syndrome $\mathbf{s}$ of the received vector determines the syndrome of the error pattern $\mathbf{e}$. Therefore, in syndrome decoding the task becomes finding the most likely error pattern within the coset indexed by $\mathbf{s}$. As shown in [3], starting from the correlation metric,

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \sum_{i=1}^{n} x_i L_i^{\text{ch}}, \tag{1.15}$$

each term $x_i L_i^{\text{ch}}$ can be expressed as:

$$x_i L_i^{\text{ch}} = \begin{cases} |L_i^{\text{ch}}|, & \text{if } x_i = \text{sgn}(L_i^{\text{ch}}), \\ -|L_i^{\text{ch}}|, & \text{if } x_i \neq \text{sgn}(L_i^{\text{ch}}). \end{cases}$$

Equivalently, using the binary error pattern, we can write:

$$x_i L_i^{\text{ch}} = (1 - 2e_i)\,|L_i^{\text{ch}}|. \tag{1.16}$$

Therefore, the correlation metric becomes:

$$\sum_{i=1}^{n} x_i L_i^{\text{ch}} = \sum_{i=1}^{n} (1 - 2e_i)|L_i^{\text{ch}}| = \sum_{i=1}^{n} |L_i^{\text{ch}}| - 2 \sum_{i:e_i=1} |L_i^{\text{ch}}|. \tag{1.17}$$

Since $\sum_{i=1}^{n} |L_i^{\text{ch}}|$ is independent of $\mathbf{e}$, maximizing the correlation metric is equivalent to minimizing the reliability weight of the error pattern $\sum_{i:e_i=1} |L_i^{\text{ch}}|$. The most likely error pattern is therefore the one that flips the least reliable bits, i.e., those with the smallest $|L_i^{\text{ch}}|$, under the condition that the resulting codeword $\hat{\mathbf{c}} = \mathbf{z} - \hat{\mathbf{e}}$ is a valid one which occurs only if $\hat{\mathbf{c}}\mathbf{H}^\top = 0 \iff \mathbf{s} = \mathbf{z}\mathbf{H}^\top = \hat{\mathbf{e}}\mathbf{H}^\top$. Consequently, the ML decoding rule can be written as:

$$\hat{\mathbf{e}} = \arg \min_{\substack{\mathbf{e} \in \{0,1\}^n \\ \mathbf{e}\mathbf{H}^\top = \mathbf{s}}} \sum_{i:e_i=1} |L_i^{ch}|, \tag{1.18}$$

The decoded codeword is then obtained by correcting $\mathbf{z}$ using the most likely error pattern $\hat{\mathbf{e}}$ as $\hat{\mathbf{c}} = \mathbf{z} - \hat{\mathbf{e}}$. Since arithmetic is performed over $\mathbb{F}_2$, this subtraction is equivalent to bitwise XOR: $\hat{\mathbf{c}} = \mathbf{z} \oplus \hat{\mathbf{e}}$.

1.2.3 Complexity of MLD

Both the correlation-based and syndrome-based formulations involve searching over a space of size 2^k. In the correlation-based formulation, the decoder must evaluate the correlation metric for all 2^k possible codewords in the code $\mathcal{C}$. In the syndrome-based formulation, there are 2^{n-k} syndrome cosets, each containing 2^k vectors. Since the syndrome uniquely identifies a coset, decoding reduces to searching within that coset. This means that the number of candidate error patterns to consider for each syndrome is also 2^k. Therefore, both the correlation-based and syndrome-based formulations have exponential complexity as a function of the message length k.

Brute-force implementation of MLD becomes computationally infeasible as the dimension k of the code increases. Ordered Statistics Decoding (OSD) [4] offers a practical alternative that significantly reduces the decoding complexity while closely approaching MLD performance. Based on information set decoding, OSD leverages soft reliability information from the channel to infer the most likely transmitted codeword. However the worst case computational complexity of OSD is dominated by the number of candidate error patterns to re-encode. While significantly lower than the full 2^k codeword search required by MLD, the total number of candidates still grows quickly with the code dimension k and order p (maximum number of bit flips performed within the most reliable independent bits in the received codeword). Therefore, in practice, small values of p are used to balance decoding performance with computational cost. It is recommended that p should be set to $\lceil d_{\min}/4 \rceil$, in order to achieve near-MLD performance for short to moderate block length codes [4].

While MLD and OSD serve as performance benchmarks, their complexity makes them impractical for many real-world systems. This limitation has motivated the search for other decoding approaches, including model-free neural decoders. Among model-free neural decoders, SBND decoders have demonstrated the best performance so far.

1.3 Principle of syndrome-based neural decoding

This section introduces the general framework of SBND, where a NN is trained as a noise estimation function to infer the most likely error pattern affecting the transmitted codeword. We outline the main training procedure, including the construction of input features, the definition of target labels, and the choice of loss function.

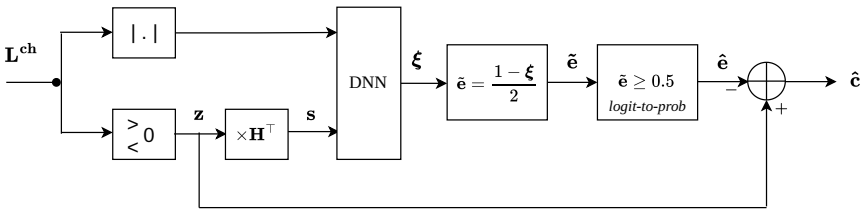


Figure 1.2: General architecture of a syndrome-based neural decoder.

1.3.1 Principle

Recall that the syndrome-based formulation of MLD (1.18) shows that the pair $(|\mathbf{L}^{\text{ch}}|, \mathbf{s})$ provides sufficient statistics to infer the most likely error pattern within the coset indexed by the syndrome $\mathbf{s}$. Building on this idea, and as illustrated in Fig. 1.2, SBND aims to estimate the most likely binary error pattern $\hat{\mathbf{e}}$ in the coset corresponding to the hard decision $\mathbf{z}$ on the received word. This estimation is carried out via a noise estimation function, typically modeled by a deep neural network (DNN). Once the estimated bianry error pattern $\hat{\mathbf{e}}$ is obtained, an estimate of the transmitted codeword is easily recovered by correcting $\mathbf{z}$, i.e., $\hat{\mathbf{c}} = \mathbf{z} - \hat{\mathbf{e}}$. In other words, the bits identified as erroneous by the DNN are flipped to reconstruct the codeword estimate.

1.3.2 Training Method

As with other model-free neural decoders, SBND are typically trained in a supervised manner. This involves constructing labeled training examples consisting of input-output pairs derived from simulated transmissions over the channel. The input to the DNN model is a concatenation of two elements: the absolute values of the channel LLRs, $|\mathbf{L}^{\text{ch}}|$, and the syndrome vector $\mathbf{s} = \mathbf{z}\mathbf{H}^\top$, where $\mathbf{z}$ is the hard-decision vector obtained from the received word and $\mathbf{H}$ is the parity-check matrix of the code.

The target label associated with each input is the binary true error pattern $\mathbf{e} = \mathbf{z} - \mathbf{c}$, where $\mathbf{c}$ is the transmitted codeword. Since we are dealing with linear codes and as stated before, it is sufficient to assume the transmission of the all-zero codeword in order to generate the training data. Under this assumption, the error pattern simplifies to $\mathbf{e} = \mathbf{z}$. Training data is obtained through Monte Carlo simulation of the transmission model described in Section 1.2.1, where simulated transmissions of the all-zero codeword are used to construct fresh input-output pairs.

In the literature, the training data is often generated on-demand. This refers to generating fresh batches of noisy received words at every step of the training process, resulting in a dynamically evolving dataset.

As for the training process, one can formulate the learning problem as a multiclass classification problem where the goal of the model is to infer the most likely error pattern from the 2^k candidate error patterns that comprise the coset indexed by $\mathbf{s}$. Therefore, the cross-entropy (CE) loss function would appear to be the most appropriate criterion for training the DNN component. However, the exponential nature of the number of classes makes this approach computationally intractable. To overcome this challenge, the commonly adopted solution is to minimize the Binary Cross-Entropy (BCE) loss instead.

In this setting, the model outputs raw logits denoted by $\boldsymbol{\xi} \in \mathbb{R}^n$, where each element corresponds to the logit associated with a particular bit being in error. These logits are passed through a hyperbolic tangent activation to produce values in the range $(-1, 1)$. To interpret them as soft probabilities, we apply the transformation:

$$\tilde{\mathbf{e}} = \frac{1 - \boldsymbol{\xi}}{2}, \tag{1.19}$$

where $\tilde{e}_i \in (0, 1)$ represents the predicted probability that bit i is erroneous. From that the estimated

binary error pattern can be obtained by thresholding each $\tilde{e}_i$, typically at 0.5, such that:

$$\hat{e}_i = \begin{cases} 1, & \text{if } \tilde{e}_i \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \quad \text{for } i = 1, \ldots, n, \tag{1.20}$$

Given the ground-truth binary error pattern $\mathbf{e} \in \{0,1\}^n$, the BCE loss is then computed as:

$$\mathcal{L}(\mathbf{e}, \tilde{\mathbf{e}}) = \frac{1}{n} \sum_{i=1}^{n} \left[-e_i \log_2 (\tilde{e}_i) - (1 - e_i) \log_2 (1 - \tilde{e}_i) \right]. \tag{1.21}$$

This approach allows the DNN to focus on learning a per-bit likelihood of error, thus converting the original intractable multiclass problem into a more manageable bitwise binary classification task.

1.4 DNN architectures for SBND

As previously discussed, model-free neural decoding benefits from the flexibility to adopt various NN architectures. SBND is no exception. The underlying decoding framework places no restrictions on the form of the noise estimation function, allowing researchers to explore a wide range of deep learning models. In what follows, we briefly highlight some of the representative architectures that have been explored in the literature.

1.4.1 Multi-layer Perceptron (MLP)

The MLP architecture was originally introduced in [1]. It follows the classical design of fully connected (FC) feedforward layers, with a key modification: the input vector is concatenated to the input of each hidden layer instead of being fed only to the first layer (see Fig. 1.3). This modification was borrowed from the behavior of a BP decoder where intrinsic information in the form of LLR is fed at each decoding iteration. The network typically consists of several hidden layers, each with a predefined size. The output layer is set to have dimension n, matching the length of the code, to allow the network to output an estimate of the binary error pattern.

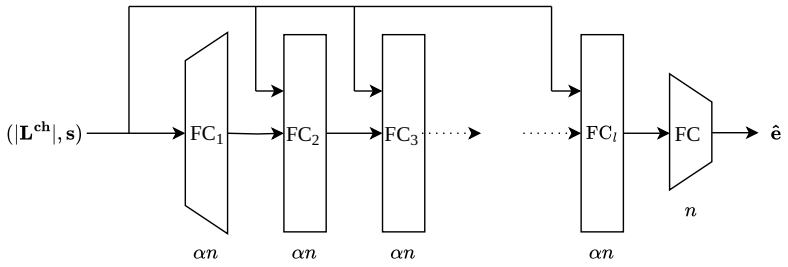


Figure 1.3: MLP-Based DNN architecture as proposed in [1]. There are l hidden layers each of size αn while the output layer is of size n

1.4.2 Recurrent Neural Network (RNN)

Originally proposed in [1], the recurrent neural network (RNN) architecture consists of multiple stacked Gated Recurrent Units (GRUs) [5] operating across timesteps. The GRU is a type of RNN that uses gating mechanisms to selectively update the hidden state at each time step allowing them to remember important information while discarding irrelevant details. A typical structure of a GRU is depicted in Fig.1.4. Although RNNs are typically designed for sequential data, here the input, comprising the magnitude vector $|\mathbf{L}^{\text{ch}}|$ and syndrome $\mathbf{s}$, is repeated at each timestep. The model is characterized by three hyperparameters: the hidden size h that governs the model's representational capacity, the

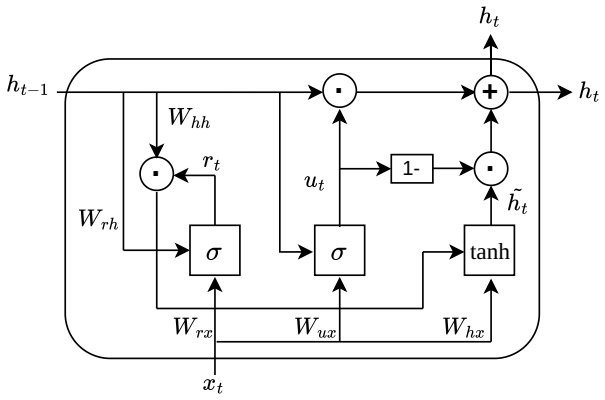
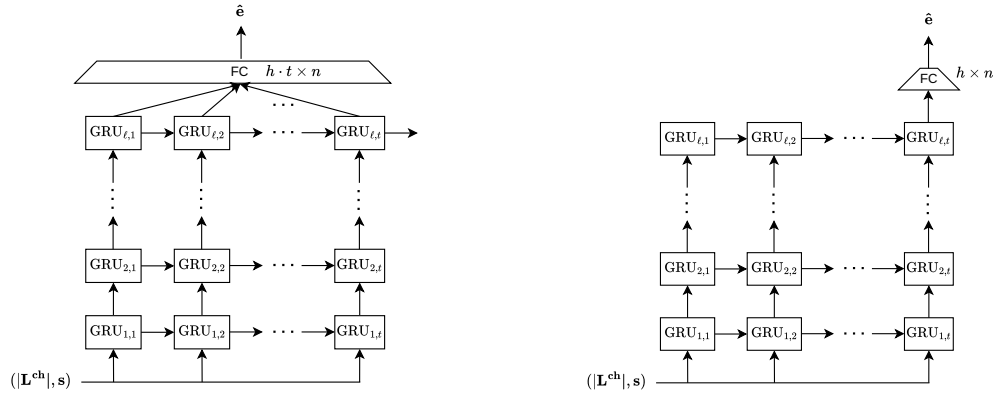


Figure 1.4: A typical structure of a GRU.

number of layers ℓ which has the effect of deepening the model, and the number of timesteps t_s which governs the temporal dimensions.

To produce the final error pattern estimate $\hat{\mathbf{e}}$, some implementations apply a FC layer of size n to the concatenated vector of the outputs at all timesteps [6] as shown in Fig. 1.4.2, while others [7] use only the output from the final timestep as input to a FC layer of size n, as illustrated in Fig. 1.4.2.



(a) (b)

Figure 1.5: A stacked GRU architecture followed by a fully connected layers (a) as in [6] (b) as in [7].

1.4.3 Transformer

Originally proposed in [2], the transformer-based architecture known as the Error Correction Code Transformer (ECCT) was introduced as the core DNN for SBND models. ECCT, adapted from the original Transformer architecture in [8], relies on self-attention mechanisms to capture global dependencies in the input. It is characterized by three main hyperparameters: the embedding dimension d, the number of attention layers ℓ, and the number of self-attention heads a. As shown in Fig. 1.6, the architecture consists of three main stages: (i) *Embedding*, where the input pair $(|\mathbf{L}^{\mathrm{ch}}|, \mathbf{s})$ is projected into a d-dimensional space, producing a sequence of $(2n-k)$ embedding vectors; (ii) *Decoding*, repeated ℓ times, consists of multi-head self-attention (MH-SA) with a heads and feed-forward sublayers, each with pre-normalization layers and followed by residual connections; and (iii) *Output*, where the final representations are processed by fully connected layers to produce the estimated error pattern $\hat{\mathbf{e}}$. A key component of ECCT is the attention mask, constructed as an extended adjacency matrix based on the parity check matrix of the code. This enables the attention mechanism to relate bits beyond

the scope of individual parity check equations, thereby capturing rich structural dependencies. We refer the reader to [2] for a detailed description of the mask construction.

More recently, [9] introduced an improved version of ECCT, called CrossMPT, which replaces the original self-attention mechanism with a cross-attention design. This modification allows the model to process the reliability and syndrome information separately while reducing the computational and memory footprint. Separately, [10] proposed a general-purpose transformer-based decoder, called FECCT, which operates across different codes and block lengths by introducing a universal representation of codewords and their structures, essentially by revisiting the embedding stage and making the mask learnable.

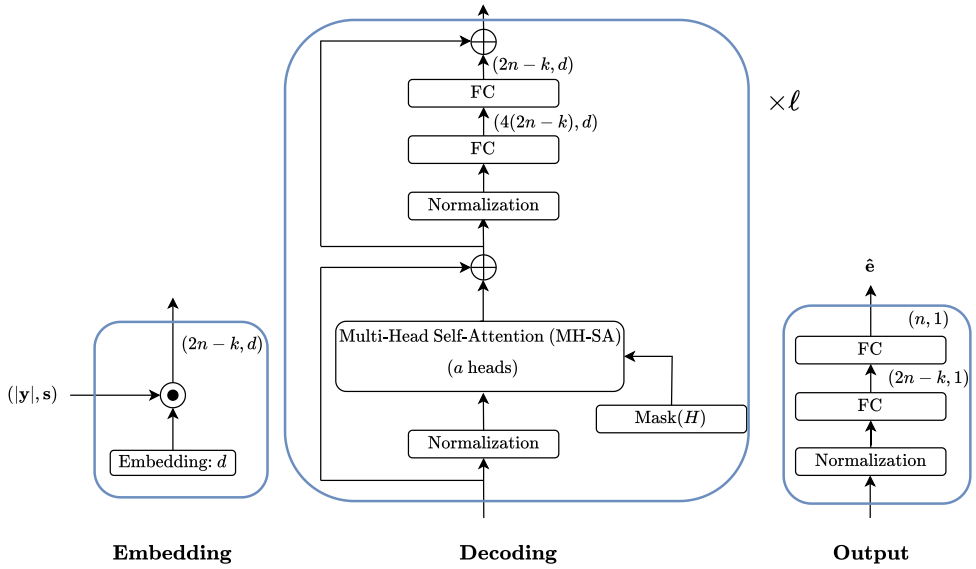


Figure 1.6: Transformer-Based DNN architecture as proposed in [2].

1.5 Performance analysis of syndrome-based neural decoders

In this section, we analyze the performance of existing SBNDs implemented using different DNN architectures. Our focus is primarily on stacked GRU and transformer-based models, as these architectures have attracted significant attention in the literature and demonstrated superior performance compared to standard MLP-based designs [1]. For evaluation, we consider two BCH codes: the $t = 2$, BCH$(31, 21)$ code with minimum distance $d_{\min} = 5$ and the $t = 3$, BCH$(63, 45)$ code with $d_{\min} = 7$. The following subsections provide details on the implementation followed by the performance analysis.

1.5.1 Implementation Setup

In what follows we provide details on the implementation setup to train SBND models in a supervised manner.

Model Input

The input to the DNN is the pair $(|\mathbf{L}^{\text{ch}}|, \mathbf{s})$. Since any scaled version of the LLR provides sufficient statistics for decoding, we replace $|\mathbf{L}^{\text{ch}}|$ with the scaled reliability vector $|\mathbf{y}|$. To ensure numerical stability and maintain a consistent input scale, $|\mathbf{y}|$ is normalized by its maximum entry:

$$\tilde{\mathbf{y}} = \frac{|\mathbf{y}|}{\max(|\mathbf{y}|)} \in [0, 1]^n. \tag{1.22}$$

This normalization preserves the relative ordering of reliabilities and therefore does not affect the MLD decision rule. In addition, the syndrome $\mathbf{s} \in \{0,1\}^m$ is represented in bipolar form as $\tilde{\mathbf{s}} = (-1)^{\mathbf{s}} \in \{1,-1\}^m$, a representation reported in prior works [2, 7] and observed during our studies to slightly improve training performance. The final model input is therefore the concatenated vector: $(\tilde{\mathbf{y}}, \tilde{\mathbf{s}})$.

DNN Implementation

We consider two widely studied DNN architectures for SBND: a stacked GRU and a transformer-based model. Their implementations are summarized as follows:

- **GRU**(ℓ, t_s): This model consists of a stacked GRU with ℓ layers and t_s timesteps. Following the approach in [7], the error pattern estimate is obtained by applying a FC layer to the output of the final timestep (see Fig. 1.4.2). The hidden size is fixed to $h = 6(2n-k)$, as recommended in [7]. Our implementation uses PyTorch's GRU module. We have noticed during our studies that the bias terms of the GRU are not necessary which allowed to reduce the number of trainable parameters.

- **ECCT**(d, ℓ): This model adopts the transformer-based ECCT architecture from [2], with embedding dimension d and ℓ layers. The number of self-attention heads is fixed to $a = 8$. We use the official source code from [11] for implementation.

Model Output

To complement the supervised learning setup, we construct the target labels $\mathbf{e}$ by applying a hard-decision rule to the received vector $\mathbf{y}$, under the common assumption that the all-zero codeword was transmitted as stated previously. During inference, the model outputs an estimate of the binary error pattern $\hat{\mathbf{e}} \in \{0,1\}^n$, which is subsequently used to recover the transmitted codeword.

Model Training

At each training iteration, some batch of noisy realizations of the all-zero codeword is generated. From these, we construct the input-output pairs $(\tilde{\mathbf{y}}, \tilde{\mathbf{s}})$ along with the corresponding binary true error pattern $\mathbf{e}$, which serves as the target label. The training objective is to optimize the model parameters by minimizing the BCE loss between the predicted error pattern $\tilde{\mathbf{e}}$ in its probability estimate format and the true error pattern $\mathbf{e}$ as defined in equation (1.21). The training hyperparameters are summarized in Table 1.1. These parameters were selected to achieve representative performance that is competitive with, and in some cases exceeds, results reported in the literature. However, no extensive hyperparameter tuning or sweep was performed.

Hyperparameter	**GRU(5,3)/ECCT(64,6)-BCH(31,21)**	**GRU(5,5)/ECCT(128,6)-BCH(63,45)**
Initial learning rate	10^{-3}	10^{-3}
LR scheduler	Cosine decay	Reduce-on-plateau
Final learning rate	$5 \cdot 10^{-5}$	adaptive
Batch size	4096	4096
Optimizer	Adam	Adam
Training SNR	3 dB	2 dB

Table 1.1: Training hyperparameters for all model-code pairs.

We also emphasize an important detail often overlooked in previous studies: whether zero-syndrome cases are included in the training set. In our implementation, we explicitly exclude such samples, since, in practical decoder operation, only received words with non-zero syndromes are passed to the SBND for correction.

1.5.2 Performance Results

The performance of the trained models is evaluated following standard channel-coding practice by computing the bit error rate (BER) and frame error rate (FER) via Monte Carlo simulations across a range of SNR points. From a learning perspective, classification models are usually assessed using classification accuracy. If the model correctly predicts the full error pattern, then this counts as a success. Therefore, the FER can be directly linked to the accuracy of the model by FER=1-test accuracy. To ensure a comprehensive assessment, the models are tested on randomly generated codewords rather than the all-zero codeword.

BCH(31,21). For the BCH$(31, 21)$ code, we train two models: GRU$(5, 3)$ and ECCT$(64, 6)$. The stacked GRU model comprises approximately 1.7 million trainable parameters, whereas the transformer-based ECCT model has around 300k parameters. Both models are trained up to approximately 1 billion noisy samples. Fig. 1.7 presents the FER and BER performance of SBND decoders using these models, compared to an order-2 OSD baseline, which is virtually identical to MLD performance on these codes.

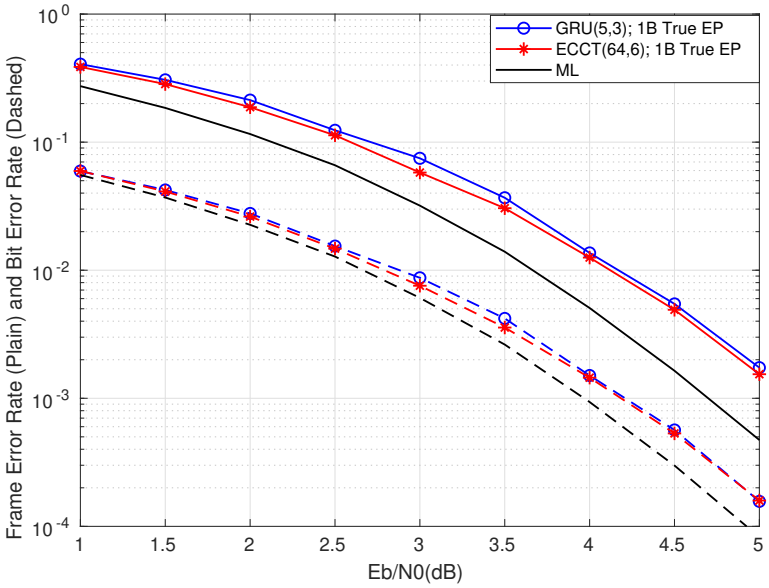


Figure 1.7: FER and BER performance curves of the different models trained on the BCH$(31, 21)$ code and compared to MLD.

BCH(63,45). For the BCH$(63, 45)$ code, we consider two models: GRU$(5, 5)$ and ECCT$(128, 6)$. The GRU-based model contains about 6.5 million parameters, while the transformer-based model has approximately 1.2 million. The GRU model is first trained up to 800 million samples and then extended to roughly 3 billion samples, whereas the ECCT model is trained up to 1 billion and then 3 billion samples. Their FER and BER performance, shown in Figure 1.8, is compared to MLD.

The results reported in Figures 1.7 and 1.8 illustrate the potential of SBND, particularly in terms of BER. However, the results also expose certain limitations of the approach, especially in the FER curves. It should also be noted that although the models are trained on a single SNR point, they demonstrate reasonable generalization across a range of SNR values.

For the BCH$(31, 21)$ code, Figure 1.7 shows that both the stacked-GRU and ECCT models achieve comparable performance, with a gap of approximately 0.22 dB from MLD in terms of BER and 0.48 dB in terms of FER at an error rate of 10^{-3}. For the BCH$(63, 45)$ code, Figure 1.8 demonstrates a gap of about 0.10 dB for BER and 0.28 dB for FER at the same error rate from the best trained model. The results reported for the GRU-based models show comparable performances with the results of [1]

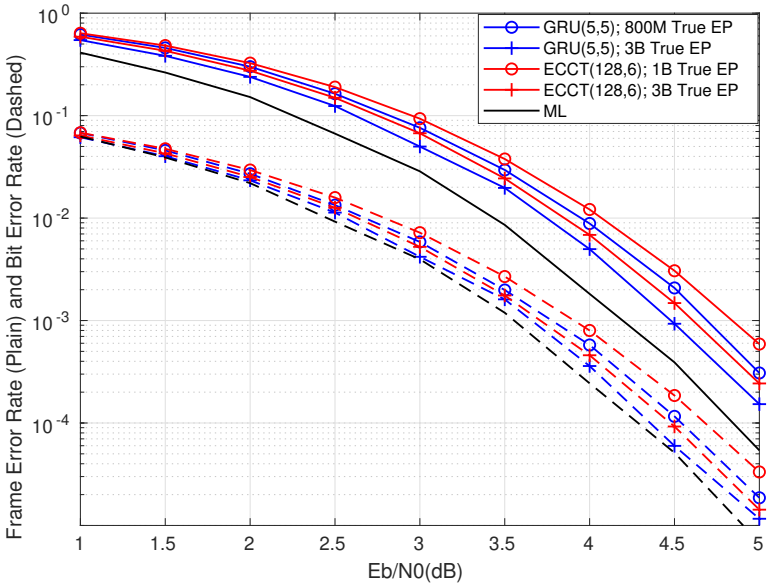


Figure 1.8: FER and BER performance curves of the different models trained on the BCH(63, 45) code and compared to the order-2 OSD for this code.

and [12]. In addition, the results of the ECCT-based model on this code outperform significantly the results reported in [2] due to training with almost 8× more samples.

It is worth noting that, for the BCH(31, 21) code, the ECCT model achieves performance comparable to the stacked-GRU while using nearly 5.5× fewer parameters. However, for the longer BCH(63, 45) code, the ECCT model lags behind the GRU-based architecture. There is no clear conclusion on whether one architecture is superior to the other, especially given that we haven't made an extensive hyperparameter study.

1.5.3 SBND limitations

Scalability. Scaling SBND to longer and lower-rate codes introduces significant challenges. Compared to BCH(31, 21), the BCH(63, 45) code not only has a larger block length but also a much larger syndrome space, resulting in increased decoding complexity. Consequently, the model requires higher representational capacity (more trainable parameters), and more training samples. These requirements grow rapidly with block length and code dimension, forming a practical bottleneck for deploying SBND on large codes.

Model size. A notable finding from the experiments is that SBND models can be quite large in terms of the number of trainable parameters, even for shorter codes. For instance, the GRU-based architecture used for BCH(31, 21) already contains approximately 1.7 million parameters. The model size depends strongly on the architecture, and both GRU-based and ECCT-based models tend to scale rapidly with code length, which poses significant challenges in terms of computational and storage requirements.

Training data size. Achieving near-MLD performance with SBND models at least in BER performance requires training on a massive number of on-demand generated samples, on the order of 10^9 in our experiments. This substantial data requirement highlights a lack of data efficiency in current SBND approaches and raises concerns about the practicality of training such models.

Residual gap to MLD. Despite extensive training, SBND models do not fully close the gap to MLD performance, particularly in terms of FER. Increasing the training dataset size initially improves

performance, but with diminishing returns. As shown in Fig. 1.9, both GRU$(5, 3)$ and ECCT$(64, 6)$ exhibit a performance plateau at SNR $= 3$ dB, where additional training yields negligible gains, and the FER asymptotically remains above the MLD benchmark. This plateau can be reduced by using larger, more expressive models, though improvements also saturate beyond a certain point. Notably, others have concluded similarly that some simple OSD decoders have been shown to outperform transformer-based SBND models sometimes significantly for certain codes [13].

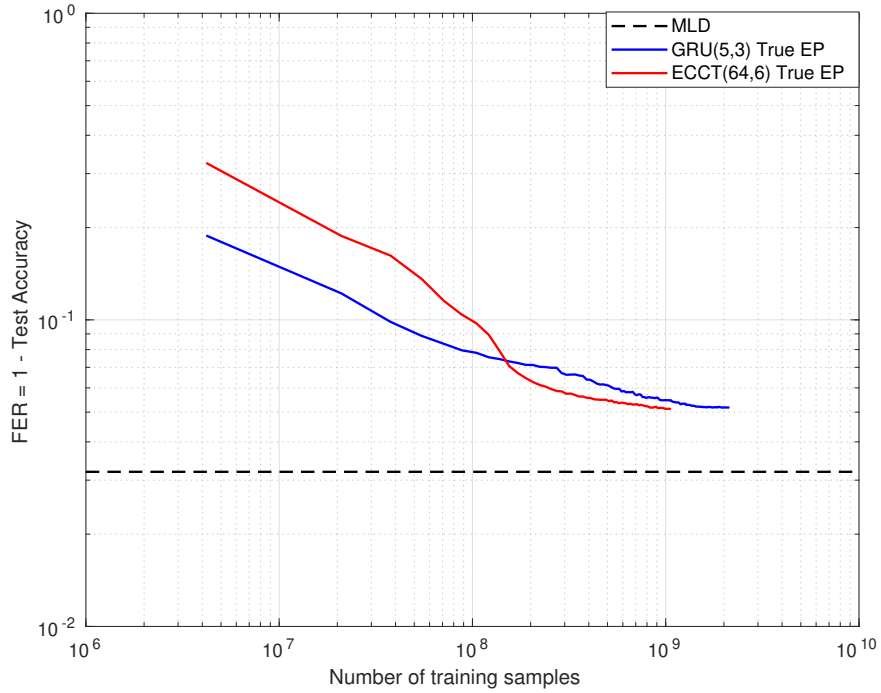


Figure 1.9: FER as a function of the number of training samples for different SBND models on the BCH(31,21) code, at SNR $= 3$ dB.

1.5.4 Summary

We introduced the principle of SBND and detailed its implementation within a supervised training framework. We reviewed existing neural decoding architectures, with particular emphasis on two prominent designs: a stacked GRU and a transformer-based model. **We have evaluated these models on various BCH codes**, demonstrating that **while they can approach near-MLD performance in terms of BER, a noticeable FER gap remains**. This holds true even after large-scale training with billions of on-demand samples, indicating significant room for improvement. **We also highlighted the resource demands of this approach**, showing that it can require surprisingly large models (over 1M parameters) and heavy training data budgets, even for short codes, raising questions about scalability. Several factors may contribute to the residual gap, including **limitations of the model architectures**, **insufficient diversity or informativeness of the training data**, and **possible shortcomings in the training methodology**. In the next section, we place special focus on the latter two aspects.

1.6 Improving training of SBND Models

This section presents the first contribution toward enhancing the performance of SBND models, with a particular focus on improving training efficiency and effectiveness. While recent research on SBND has primarily focused on designing more capable architectures, comparatively little attention has been

paid to the quality and structure of the training data and methodology. This is particularly concerning given that, as shown in the previous chapter, even for a relatively short BCH$(31, 21)$ code, achieving competitive performance required training on over one billion samples, which made us question whether this is necessary. Motivated by this, we revisit the training data and methodology commonly adopted in the literature. Specifically, we begin by identifying a mismatch between the current training targets and the actual decoding objective, which can limit performance. We then propose the use of fixed datasets as a more sample-efficient alternative to the traditional on-demand data generation process. Furthermore, we introduce targeted sampling strategies that intend to concentrate the training distribution on the most relevant examples allowing the model to learn more efficiently. Finally, we leverage the automorphism group of the code to perform data augmentation, improving generalization while maintaining efficiency.

1.6.1 Training for MLD

Current SBND models are trained for *zero-error* decoding. As discussed in Section 1.3.2, the target labels used in training are the true binary error patterns. If the transmitted codeword is $\mathbf{c}$ and the hard decision on the received word $\mathbf{y}$ is $\mathbf{z}$, the corresponding true error pattern is obtained as $\mathbf{e}_{\text{true}} = \mathbf{z} - \mathbf{c}$. This means that the models are trained to exactly correct all channel-induced errors. However, in reality, no decoder, including MLD, can ensure *zero-error* decoding. As there is no guarantee that such an algorithm actually exists, training SBND models to meet such an unattainable objective can fundamentally limit their performance.

Since the decoding objective is ultimately to approximate the MLD rule, we argue that the models should be trained to reproduce MLD behavior, which is deterministic for a given $\mathbf{y}$. This leads us to replace the true error patterns with what we call *ML error patterns*. These are defined by $\mathbf{e}_{\text{ML}} = \mathbf{z} - \mathbf{c}_{\text{ML}}$, where $\mathbf{c}_{\text{ML}}$ is the MLD decision for the received word $\mathbf{y}$, as shown in Fig. 1.10.

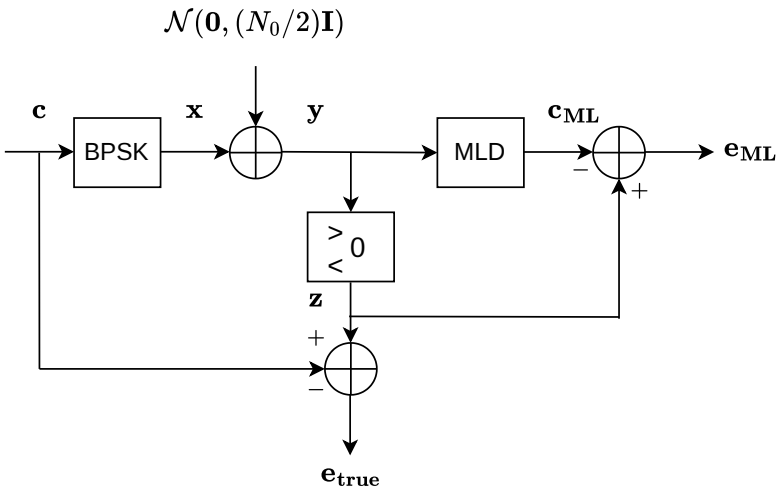


Figure 1.10: ML error patterns vs. true error patterns.

To enable training on $\mathbf{e}_{\text{ML}}$ targets, we resort to an OSD [4]. Specifically, we employ an OSD with a maximum reprocessing order $p = \lfloor d_{\min}/4 \rfloor$, which have shown to provide near-MLD performance for the short codes considered. Training data are generated via Monte Carlo simulation of the channel: for each received vector, OSD produces the most likely codeword estimate $\mathbf{c}_{\text{ML}}$, from which we infer $\mathbf{e}_{\text{ML}}$, serving as the training target labels.

1.6.2 Training with fixed datasets

As discussed earlier, SBND models are typically trained using an on-demand data generation process, where a fresh batch of noisy realizations is produced at each training step to form the input-output pairs. This approach is in contrast to the conventional deep learning approach of using fixed datasets. With fixed datasets, a finite set of training examples is generated once and reused throughout training. During each epoch, the model iterates over the same examples, typically shuffled in a different order.

While the on-demand data generation approach avoids the extensive memory requirements associated with storing large training sets, it is not sample-efficient. In practice, models often require billions of samples to achieve a sufficiently good performance, even for relatively short codes. On the other hand, with fixed datasets, optimizers like stochastic gradient descent (SGD) and its variants tend to converge faster. The intuition is that repeated exposure to the same data allows the model to incrementally refine its predictions. This leads to more stable gradient updates with reduced variance, which ultimately contributes to faster convergence. This suggests that fixed datasets could be a more sample-efficient approach.

In addition, adopting fixed datasets offers several other advantages. First, it provides full control over the size, nature, and distribution of the generated data. This is particularly crucial when working with ML error patterns: as explained in the previous section, generating these patterns requires OSD decoding, which is prohibitively expensive if performed online at every training step. By pre-computing ML error patterns and storing them in a fixed dataset, we avoid this computational bottleneck. Moreover, fixed datasets enable reproducibility and fair comparison. Once generated, they can be shared publicly, allowing researchers to benchmark their models under identical conditions.[1]

We intend to confirm whether training models on fixed datasets is more sample-efficient than training on-demand. Then, with the help of fixed datasets, we wish to asses whether training with ML error patterns improve the model's performance. To this end, we consider again the two BCH codes, namely the BCH$(31, 21, d_{\min} = 5)$ and BCH$(63, 45, d_{\min} = 7)$ and we employ the same model architectures introduced in Section 1.5.1: GRU(5,3) and ECCT(64,6) models for the BCH$(31, 21, 5)$ code, and the GRU(5,5) and ECCT(128,6) models for the BCH$(63, 45, 7)$ code. We train these models on fixed datasets, either with true error patterns of ML error patterns. Since training on fixed datasets can eventually lead to model overfitting, we adjusted the training procedure accordingly. The new hyperparameters are now summarized in table 1.2. Overfitting occurs when the model excels in preforming predictions on the training samples but fails miserably to generalize to unseen ones. This happens because the model has memorized the training set, learning noise and irrelevant patterns rather than capturing the underlying mapping rule. This leaves the model incapable of performing well on new test data, even if it differs only slightly from the training data.

Hyperparameter	**GRU(5,3)-BCH(31,21)**	**ECCT(64,6)-BCH(31,21)**	**GRU(5,5)-BCH(63,45)**	**ECCT(128,6)-BCH(63,45)**
Epochs	256	256	256	128
Batch size	4096	4096	4096	4096
Optimizer	AdamW	AdamW	AdamW	AdamW
Dropout	0.2	0.01	0.2	0.01
Initial learning rate	0.001	0.001	0.0005	0.0006
Weight decay	0.02	None	0.02	0.01
Training SNR	3 dB	3 dB	2 dB	2 dB

Table 1.2: Training hyperparameters for all model-code pairs.

Should one train on demand or with fixed datasets?

Fig. 1.6.2 presents the performance of the GRU(5,3) model trained on a fixed dataset of 4M true error patterns for the BCH$(31, 21, 5)$ code. Its performance is on par with, and even slightly surpasses, the model discussed in section 1.5, which was trained on 1B on-demand generated error patterns. This represents 250× reduction in training samples. Similarly, for the BCH$(63, 45, 7)$ code in Fig. 1.6.2,

[1]Some of the datasets created for this study are available on the AI4CODE project homepage https://ai4code.projects.labsticc.fr/.

training with a fixed dataset of 100M samples achieves comparable results to when training on 800M on-demand samples, corresponding to an 8× reduction in sample size. This highlights that training with fixed datasets is substantially more sample-efficient than training with on-demand data, confirming that models indeed benefit from seeing the same examples several times during training.

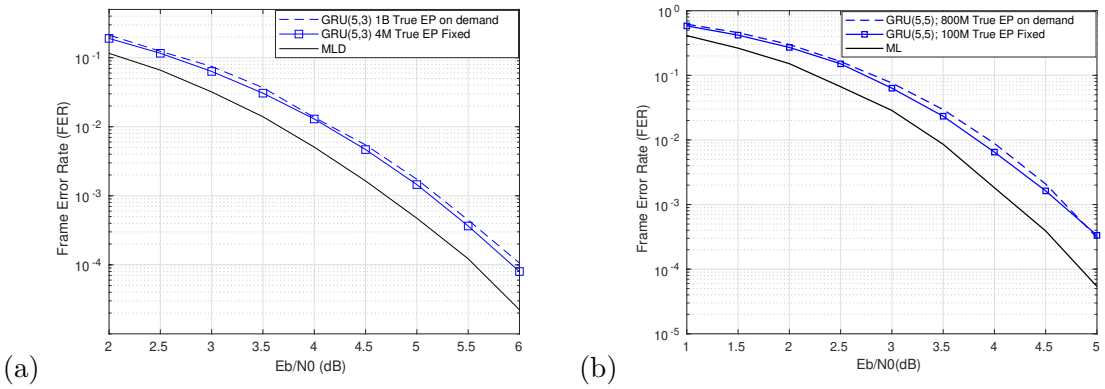


Figure 1.11: FER performance of GRU models trained using true error patterns either on-demand or with a fixed dataset: (a) GRU(5,3) on BCH(31, 21, 5), (b) GRU(5,5) on BCH(63, 45, 7).

Should one learn to correct MLD or true error patterns?

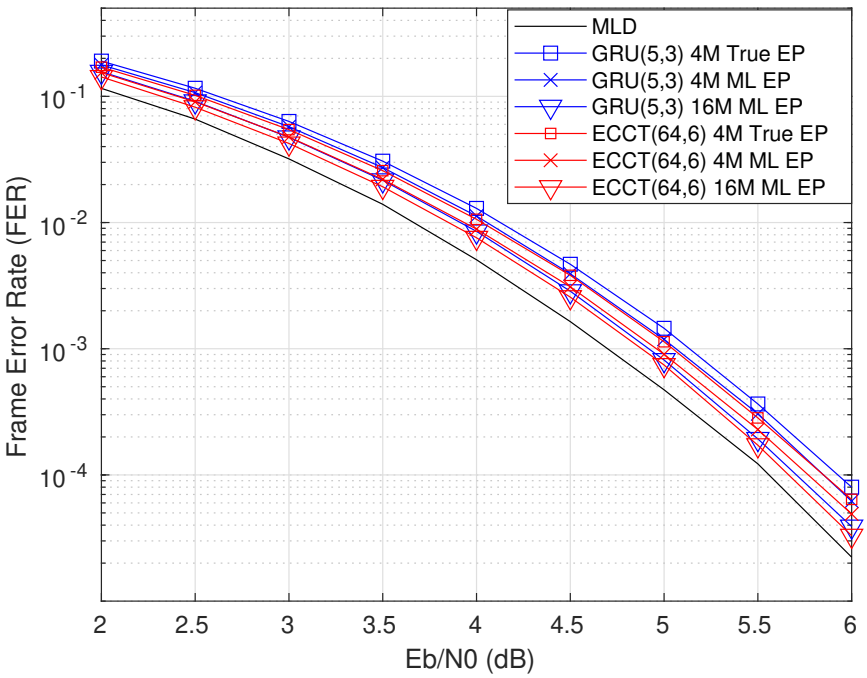


Figure 1.12: FER performance of different SBND models trained on true vs. ML error patterns for the BCH(31, 21, 5) code.

In Fig. 1.12, we compare the performance of the GRU(5,3) and ECCT(64,6) models on the BCH(31, 21, 5) code when trained on fixed datasets using either true error patterns or ML error patterns. The results show that models trained on ML error patterns consistently outperform those trained on true error patterns across all dataset sizes. A similar trend is observed for the BCH(63, 45, 7) code in Fig. 1.13. Both the GRU-based and the transformer-based models outperformed their counterparts trained on true error patterns. Note that the ECCT(128,6) trained on a fixed dataset of 64M

true error patterns differ slightly in terms of hyperparmeters compared to the one reported in table 1.2. This includes a slightly larger learning rate of 0.0008 and no weight decay factor.

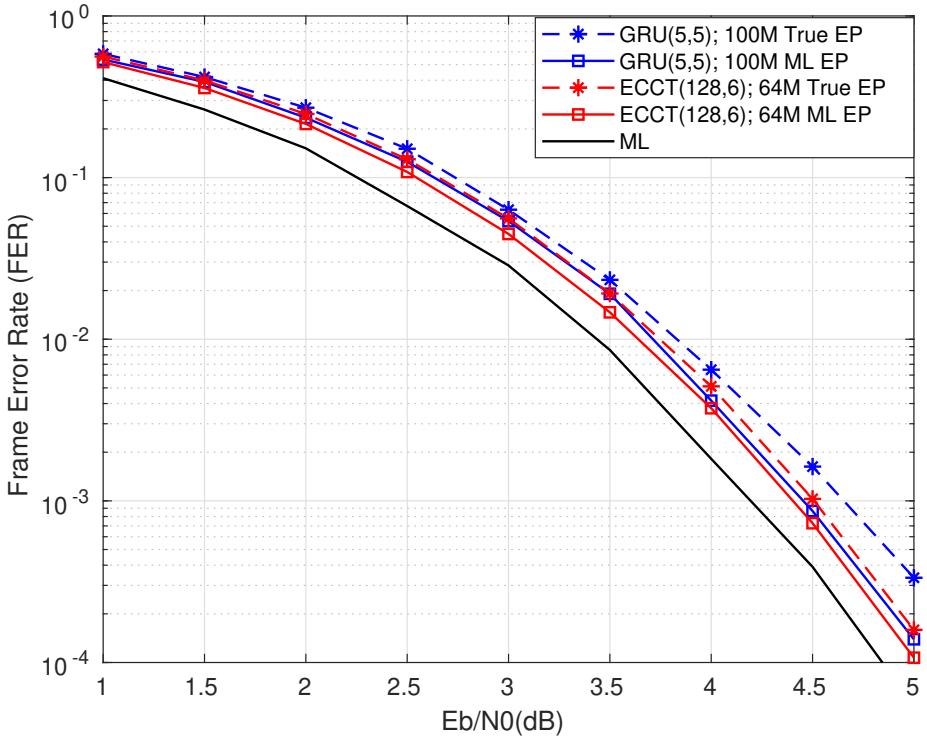


Figure 1.13: FER performance of different SBND models trained on true vs. ML error patterns for the BCH$(63, 45, 7)$ code.

These findings suggest that training on ML error patterns allow models to generalize better. It is important to note that the difference between ML and true error pattern datasets arises only in instances where MLD makes a decoding error. To effectively train models on ML error patterns, one should select lower SNR points, where this difference is more pronounced. Nevertheless, the chosen SNR values for both codes in these experiments appear sufficient to highlight the benefits of ML-based training.

Performance limit of architectures and models

Fig. 1.14 shows the FER as a function of the number of training samples for the BCH$(31, 21)$ code at SNR $= 3$ dB, now including models trained on fixed datasets. The results reinforce the conclusion that training on fixed datasets is significantly more sample-efficient compared to training on-demand. Moreover, the figure clearly illustrates that models trained on ML error patterns consistently outperform those trained on true error patterns, across both the GRU(5,3) and ECCT(64,6) architectures. This advantage is especially pronounced for smaller dataset sizes.

As observed in earlier chapters, the performance of these models approaches an asymptotic plateau beyond which additional training data brings diminishing returns. A similar saturation effect is evident when training on fixed datasets. For example, for this code, increasing the size of the dataset beyond approximately 16M ML error patterns yields only marginal performance improvements. Although the performance was improved by training on ML error patterns, a noticeable gap to MLD is still observed. As discussed in 1.5.3, this gap can essentially be reduced when using more complex models but still up to a certain limit.

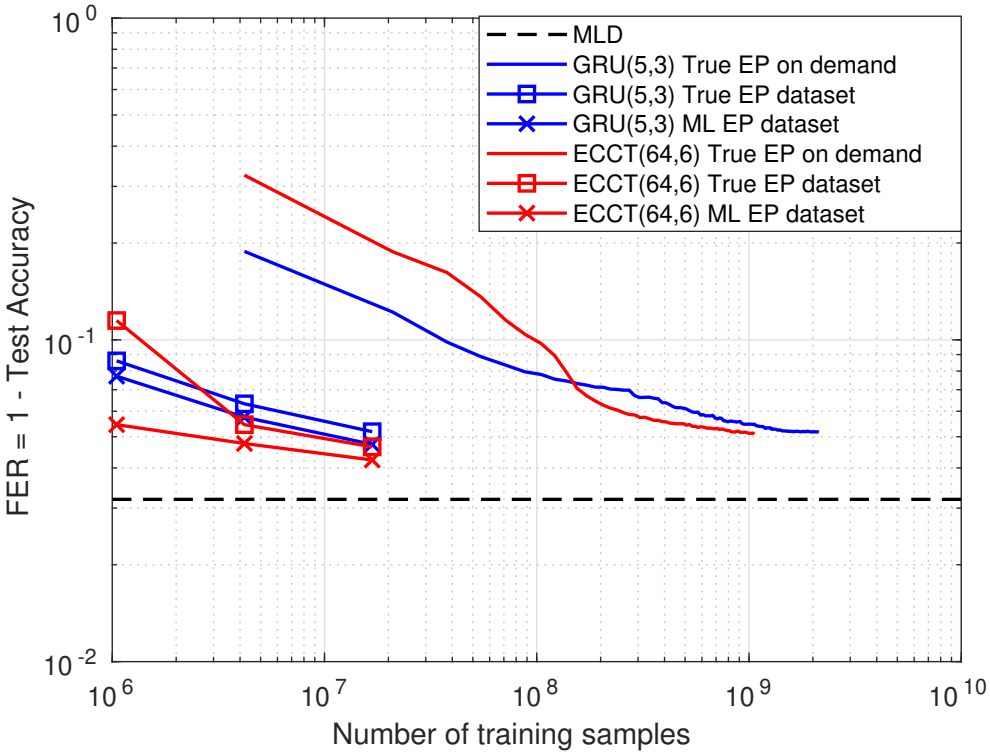


Figure 1.14: FER performance of different SBND models trained on true vs. ML error patterns for the BCH$(63, 45, 7)$ code.

1.6.3 Optimizing the training distribution

Training data for SBND models are typically generated via Monte Carlo simulation (with either true or ML error patterns), as discussed previously. While this follows standard practice in channel coding, it is inefficient for rare-event learning. The resulting datasets are dominated by samples of low-weight error patterns that are easy for the model to correct.

This imbalance originates from the nature of the AWGN channel. Specifically, the BI-AWGN channel induces a binomial distribution on the Hamming weight $w_H(\mathbf{e}_{\text{true}})$ of the true error patterns:

$$p_{\text{chan}}(w) = \binom{n}{w} p_b^w (1-p_b)^{n-w}, \quad w = 0, \ldots, n, \tag{1.23}$$

where $p_b = \frac{1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right)$ is the bit error probability. Consequently, datasets generated directly from the BI-AWGN channel are heavily skewed toward low-weight errors. By contrast, higher-weight error patterns are both underrepresented and far harder to correct, yet the model's ability to handle them largely determines its FER performance, especially at moderate to low SNR.

In the broader deep learning literature, it is well-recognized that not all training samples contribute equally to effective learning [14]. Recent research emphasizes dataset curation and sample selection strategies to prioritize the most informative examples for training in an effort to improve the model's generalization capability [15–17]. Similarly, in neural decoding, [18] was among the first to advocate for smarter sampling strategies to improve the efficiency of model-based neural BP decoders. Motivated by these insights, and with the help of fixed datasets, we have full control to reshape the training distribution for SBND models by carefully selecting the received words $\mathbf{y}$ with their associated error patterns $\mathbf{e}$ to ensure that the models are exposed to more meaningful training examples. To this end, we leverage upon the Hamming weight of the error patterns as a proxy metric to help distinguish informative and less informative samples. Accordingly, we modify the empirical weight distribution $p(w)$ of the error pattern.

Determining an optimal training distribution for neural decoding remains a difficult challenge. In the absence of well-designed guidelines, we investigate several heuristic approaches to construct the training dataset. Our goal is twofold: (1) to empirically demonstrate that using distinct distributions for training and testing can enhance data efficiency and boost model performance, and (2) to identify simple yet effective heuristics for sample selection. We explore four different methods, each relying on Monte Carlo simulations of the OSD decoder, which differ in their sample selection criteria. The first method serves as a baseline reference, while the remaining three attempt to filter out the most impactful MLD error patterns that critically influence FER. A brief description of each approach follows.

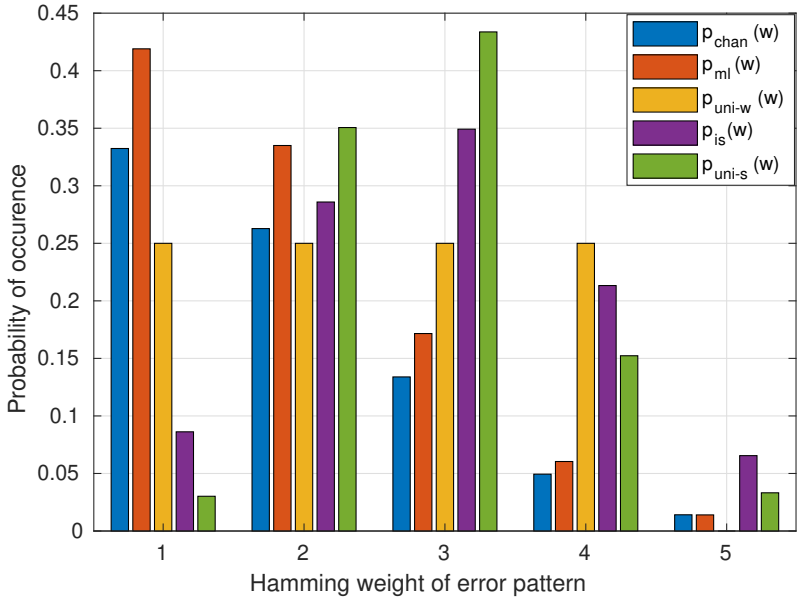


Figure 1.15: Weight distribution of error patterns obtained with different training set constructions for the (31, 21) code at SNR = 3 dB.

Method 1: Using the channel error distribution. This method serves as the baseline for comparison. As described in Section 1.6.1, training targets are derived from a Monte Carlo simulation of the AWGN channel followed by decoding with MLD. The resulting ML error patterns, $\mathbf{e}_{\text{ML}}$, are collected directly from the decoder output without applying any form of biased sampling or filtering. Since the input to the decoder reflects the binomial distribution $p_{\text{chan}}(w)$ of the AWGN-induced error weights, the output patterns inherit a transformed distribution $p_{\text{ML}}(w)$. This distribution differs a little from the original channel distribution and is typically more concentrated around lower-weight error patterns as seen in Fig. 1.15. Since MLD decodes to the nearest codeword, if MLD commits a decoding error this will change distribution from the original and explain the large low-weight patterns at it output. This is explained by the fact that MLD decodes to the nearest codeword. When this decoded codeword is incorrect, the original distribution begins to shift and becomes biased toward lower-weight error patterns that result from decoding to nearby codewords.

Method 2: Using a uniform distribution of the weights. To promote balanced learning across error weights, this method constructs a training dataset with a uniform distribution over the Hamming weights of the error patterns. First, we discard all ML error patterns $\mathbf{e}_{\text{ML}}$ whose Hamming weight exceeds a predefined threshold $w_{\max}$ (set to 4 here). From the remaining error patterns, we then sample to achieve a uniform weight distribution, denoted as $p_{\text{uni-w}}(w) = 1/w_{\max}$ for $w = 1, \ldots, w_{\max}$, within the dataset.

Method 3: Using a biased input error distribution. The training examples $\mathbf{y}$ provided to the MLD are generated according to an input distribution $p^{(\text{is})}_{\text{chan}}(w)$, which differs from the binomial distribution $p_{\text{chan}}(w)$ induced by the AWGN channel. Specifically, the noise importance sampling distribution $p^{(\text{is})}_{\text{chan}}(w)$ is designed to be optimal in terms of minimizing the number of required samples to achieve a target error rate by encouraging the occurrence of errors. It is determined following the procedure outlined in [19]. The corresponding weight distribution of error patterns at the MLD decoder output will be referred to as $p_{\text{is}}(w)$. Compared to the standard approach (method 1), this method results in a distribution that places greater emphasis on error patterns of weights 3 and 4 as illustrated in Fig. 1.15.

Method 4: Using a biased input error distribution. Since the syndrome plays a key role in the MLD decision rule and is also part of the model input, it is reasonable to ensure that all possible non-zero syndromes are well represented in the training set. A simple yet effective way to achieve this is to filter the ML error patterns collected from the output of MLD, selecting an equal number of patterns for each syndrome value. As with method 3, this naturally places more emphasis on error patterns of weights 3 and 4. This approach can also be combined with methods 2 and 3. Unlike methods 1–3, which scale with the code parameters (n, k), this fourth method requires that the total number of syndromes 2^{n-k} remains reasonably small. The resulting error weight distribution is denoted by $p_{\text{uni-s}}(w)$.

We aim to assess whether the proposed heuristics enable more efficient training of SBND models, and whether any method consistently outperforms the others. For that, we consider the BCH$(31, 21, 5)$ code with the GRU(5,3) model, trained as described in table 1.1. We construct four distinct datasets, each containing 4M samples at an SNR of 3 dB, corresponding to the four proposed sampling distributions. Using these datasets, we train the GRU(5,3) model under the same setup. During inference time, all models are tested under the same conditions as described previously: random noisy codewords generated using Monte Carlo simulations across various SNRs. The resulting FER performance, shown in Fig. 1.16, indicates that training on datasets with distributions differing from the natural BI-AWGN-induced distribution, particularly method 3, leads to models with superior performance compared to those trained with error patterns obtained from standard Monte Carlo simulation using true or MLD (method 1). Although no single method emerges as clearly optimal in this example, the results underscore the benefits of optimizing the training distribution, especially when dataset sizes are limited. The results show as well, that while training and testing distributions are distinct, the models were capable to generalize reasonably well.

For the BCH$(63, 45, 7)$ code, we trained a GRU(5,5) model following the setup described in Section 1.17, using ML error patterns sampled according to method 3. Its performance is compared against the two baselines: training on the standard dataset of 100 ML error patterns and training with 3 billion on-demand samples. Remarkably, using only a 32M-sample dataset, the model achieves slightly better performance, underscoring the value of carefully curated training data, which not only allows the models to generalize better but to do so with much fewer samples. Expanding the dataset to 64M samples yields only marginal gains (not shown here), indicating, similar to the observations for the BCH$(31, 21, 5)$ code, that the model is nearing its performance ceiling. The same figure also reports the BER of this trained model, showing that its BER is nearly equivalent to MLD, similar to the model trained on 3B true error patterns, but with almost 100× fewer training samples.

1.6.4 Taking advantage of data augmentation

Data augmentation is a widely adopted strategy in deep learning to enhance model generalization, especially when training data is limited [20]. It involves applying a set of predefined transformations to each training example, thereby increasing the diversity of data encountered during training. Data augmentation has been particularly praised for its effectiveness in classification tasks, especially image classification, where a variety of distortions, such as horizontal flipping, random cropping, color jittering, and affine transformations, can be applied. These augmentations help models become robust to

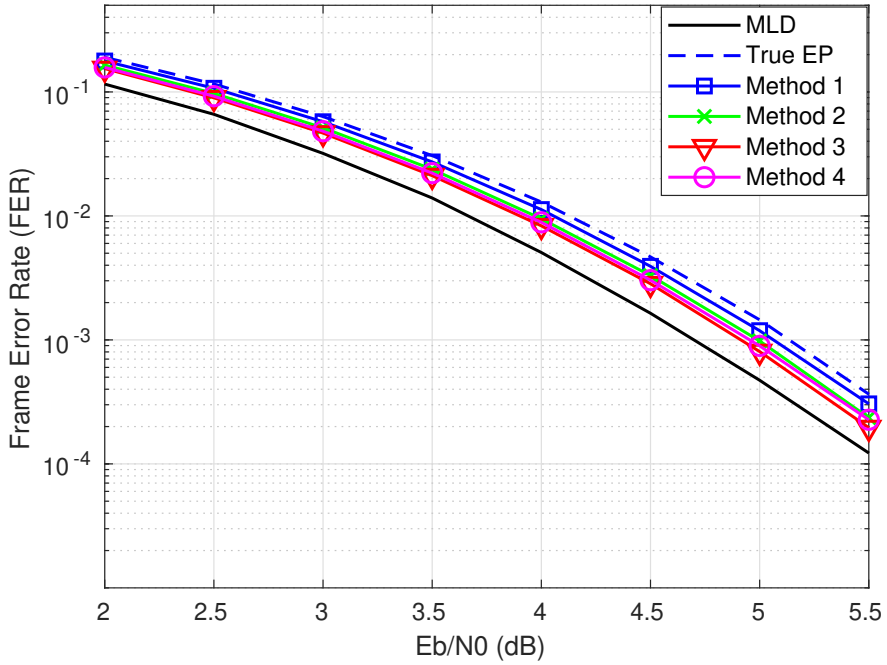


Figure 1.16: FER for a GRU(5, 3) model trained to decode the BCH(31, 21,5) code using different datasets of 4M samples.

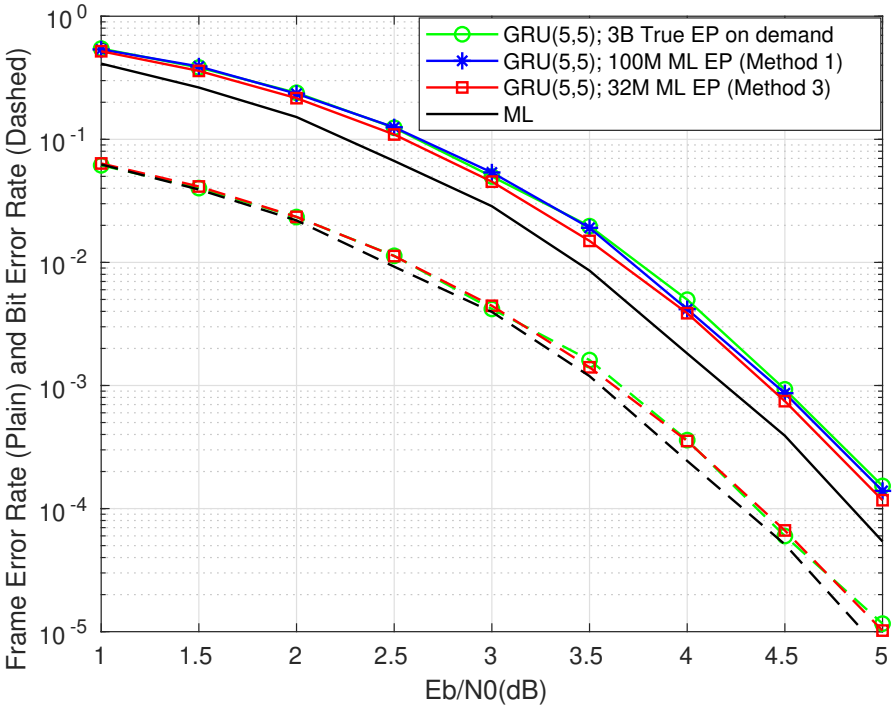


Figure 1.17: FER and BER for a GRU(5, 5) model trained to decode the BCH(63, 45,7) code using models trained with different strategies or distributions.

variations in the input, ultimately improving their generalization capabilities and consequently reducing the model's tendency to overfit. In the context of training a SBND, we propose leveraging the code automorphism group as a natural augmentation method. Because these transformations preserve the code's structure, they enable the efficient generation of new, equivalent training samples from existing

ones.

Recall that the MLD rule (1.10) selects the codeword $\mathbf{c} \in \mathcal{C}$ that minimizes the Euclidean distance to the received vector $\mathbf{y}$. Now consider applying a permutation $\pi \in \mathrm{Aut}(\mathcal{C})$, i.e., an automorphism of the code $\mathcal{C}$, to all coordinate positions. Since π maps any codeword to another valid codeword, the set $\mathcal{C}$ is invariant under π. Moreover, the Euclidean norm is also invariant under permutations, i.e.,

$$||\mathbf{y} - (-\mathbf{1})^{\mathbf{c}}||^{\mathbf{2}} = \|\pi(\mathbf{y}) - (-\mathbf{1})^{\pi(\mathbf{c})}\|^{\mathbf{2}}, \tag{1.24}$$

where $\pi(\mathbf{c}) = \big(c_{\pi^{-1}(1)}, c_{\pi^{-1}(2)}, \ldots, c_{\pi^{-1}(n)}\big)$. As a result, the ML decision does not change in substance; it simply gets permuted. That is,

$$\pi(\hat{\mathbf{c}}) = \pi\left(\arg\min_{\mathbf{c}\in\mathcal{C}} \|\mathbf{y} - (-1)^{\mathbf{c}}\|^2\right) = \arg\min_{\pi(\mathbf{c})\in\mathcal{C}} \|\pi(\mathbf{y}) - (-1)^{\pi(\mathbf{c})}\|^2, \tag{1.25}$$

which means that the ML decision for $\pi(\mathbf{y})$ is simply the permuted version of the ML decision for $\mathbf{y}$. Equivalently, if $\mathbf{z} = \mathbf{c} + \mathbf{e}$, where $\mathbf{z}$ is the hard decision on the received vector and $\mathbf{e}$ is the estimated error pattern on codeword $\mathbf{c}$, then the MLD rule (1.18) select the error pattern within the syndrome coset indexed by $\mathbf{z}$ that minimizes the sum of reliability values at the erroneous bit positions. Now, assume we apply a permutation $\pi \in \mathrm{Aut}(\mathcal{C})$ to all coordinate positions. Then,

$$\pi(\mathbf{z}) = \pi(\mathbf{c} + \hat{\mathbf{e}}) = \pi(\mathbf{c}) + \pi(\hat{\mathbf{e}}), \tag{1.26}$$

where $\pi(\mathbf{c}) \in \mathcal{C}$ by definition of a code automorphism. To estimate the error pattern on the permuted input $\pi(\mathbf{z})$, one must solve the same minimization problem but with permuted reliability values:

$$\pi(\hat{\mathbf{e}}) = \arg \min_{\substack{\mathbf{e}'\in\{0,1\}^n \\ \mathbf{e}'H^\top = \pi(\mathbf{z})H^\top}} \sum_{i:e'_i=1} |L^{\mathrm{ch}}_{\pi^{-1}(i)}|. \tag{1.27}$$

Therefore, if $\hat{\mathbf{e}}_{\mathrm{ML}}$ is the MLD error pattern corresponding to the pair $(|\mathbf{y}|, \mathbf{s})$, then for any automorphism $\pi \in \mathrm{Aut}(\mathcal{C})$, the permuted pattern $\pi(\hat{\mathbf{e}}_{\mathrm{ML}})$ is the MLD decision for the transformed input $\Big(|\pi(\mathbf{y})|,\ \mathbf{s}' = \pi(\mathbf{z})H^\top\Big)$. Thus, by permuting both the reliability vector $|\mathbf{y}|$ and the target error pattern $\hat{\mathbf{e}}_{\mathrm{ML}}$, and by recomputing the associated syndrome, code automorphisms offer a simple way to introduce diversity into the training examples presented to the neural decoder during training.

To assess the effectiveness data augmentation, we trained a GRU(5,3) model to decode the BCH$(31, 21, 5)$ code. Initially, the model was trained on three datasets containing 1, 4, and 16 million distinct examples of ML error patterns, respectively. These are constructed following method 1, that is the standard Monte Carlo ML error patterns. We then repeated the experiment with two augmented datasets of sizes 4 and 16 million examples. Both were derived from the 1-million-example dataset by applying, to each example, 4 and 16 random permutations, respectively, chosen from the 155 permutations available (5×31) for this code. This setup represents a worst-case scenario for diversity, as augmentation was performed once during dataset construction rather than the typical approach of generated the augmented examples on-the-fly during batch loading. Despite this, remarkably, the performance results reported in Fig. 1.18 show no difference compared to the ideal case of fully distinct examples.

1.6.5 Reflection on the proposed heuristics to improve the training of SBND

We have explored a set of heuristics designed to enhance the training of SBND models, focusing on both the nature and quality of the training data. By training the model to correct MLD error patterns, rather than targeting the idealized and practically unattainable true error patterns, and by carefully optimizing the training distribution, we have shown that **we can do more with less**: achieving superior performance with significantly fewer training samples. This improvement is made possible through the use of fixed datasets, which allow us to control data diversity and quality more effectively. This principle is further reinforced by our proposed data augmentation strategy based on code automorphisms, which effectively introduces diversity to the training process allowing models

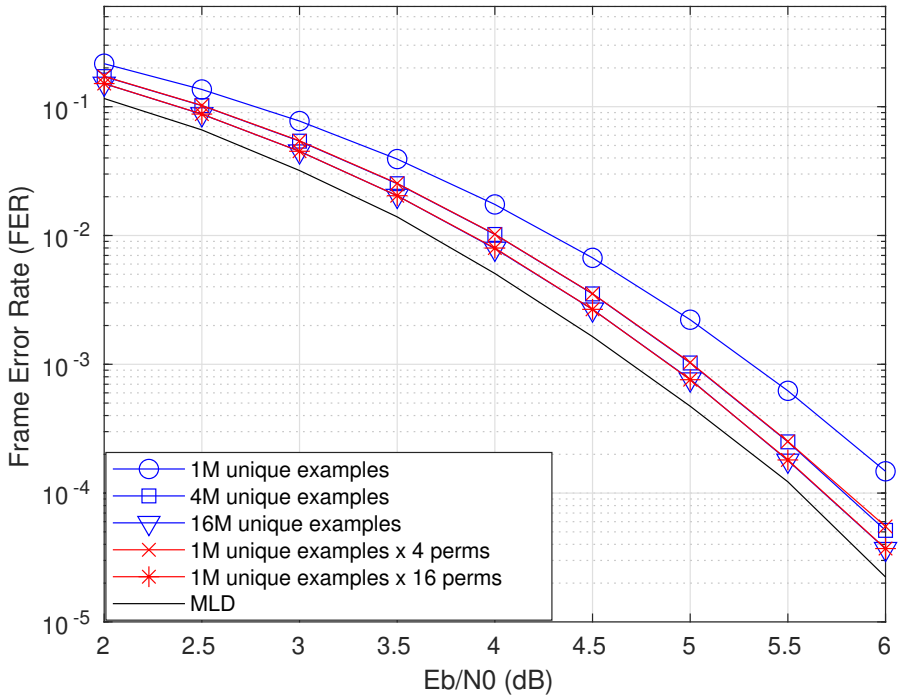


Figure 1.18: FER of a GRU(5,3) model trained on the BCH code (31,21,5) with sets of ML error patterns that are unique or augmented by automorphisms.

to generalize better on even fewer samples. Although we have not fully closed the gap to MLD, our findings emphasize that the quality and structure of the training data play a far more critical role and deserve more attention in the research on SBND. It is noteworthy that these proposed heuristic are not limited to SBND and can it turn be applied to any neural decoders.

1.7 Improving SBND performance at inference time

While careful training design is crucial for achieving strong baseline performance, further improvements can often be obtained during inference time without modifying the underlying trained model. In this section, we explore two distinct strategies to enhance the decoding capability of SBND models post-training. The first strategy is iterative correction which draws inspiration from the principle of self-boosting, leveraging multiple decoding iterations to refine the model's own predictions. The second approach employs test-time augmentation (TTA) through code automorphisms, by creating an ensemble of decoding attempts, on different inputs generated by exploiting the inherent symmetries of the code. Both of these techniques provide means to improve the performance of SBND models. In what follows, we will present the principle of both approaches and evaluate them.

1.7.1 Iterative Correction

Even after a well-trained SBND model outputs its estimate of the binary error pattern $\hat{\mathbf{e}}$, there is no guarantee that the recovered codeword $\hat{\mathbf{c}} = \mathbf{z} - \hat{\mathbf{e}}$ is a valid one. This limitation arises because SBND models are typically trained with BCE loss, which optimizes bit-wise accuracy but does not directly enforce code constraints. As a result, decoding failures may occur when the estimated error pattern does not correspond to a valid codeword. To mitigate this issue, post-processing strategies can be applied during inference to further refine the decoded output.

Two key strategies have been proposed in the literature to improve decoding success in such cases: *iterative error correction (IEC)* [21] and *iterative error decimation (IED)* [22]. Both approaches

operate by launching additional decoding attempts when the initial SBND output fails, but they differ in how they update the decoder input and refine the error estimate. These iterative methods implement sort of a self-boosting mechanism by using the model's previous outputs as feedback to progressively enhance decoding accuracy through repeated refinement. This is inspired by the boosting technique used in deep learning, where multiple weaker models are employed in an iterative fashion leveraging upon each other's predictions to improve accuracy.

Iterative Error Correction (IEC). In IEC, the SBND model is invoked iteratively to refine its estimate of the error pattern. At each iteration, the full predicted error vector is used to update the current hard decision, aiming to eventually recover a valid codeword.

Let $\mathbf{z}^{(0)}$ denote the initial hard decision on the received vector, and let $\mathbf{s}^{(0)} = \mathbf{z}^{(0)}\mathbf{H}^{\mathsf{T}}$ be the initial syndrome. IEC proceeds at iteration $j \geq 1$ as follows. The SBND model is applied to the pair $(|\mathbf{L}^{\text{ch}}|, \mathbf{s}^{(j)})$ to produce a binary error estimate $\hat{\mathbf{e}}^{(j)} \in \{0,1\}^n$. The hard decision is then updated via

$$\mathbf{z}^{(j+1)} = \mathbf{z}^{(j)} - \hat{\mathbf{e}}^{(j)}.$$

The new codeword estimate is

$$\hat{\mathbf{c}}^{(j)} = \mathbf{z}^{(j+1)},$$

and the corresponding updated syndrome is computed as

$$\mathbf{s}^{(j+1)} = \mathbf{z}^{(j+1)}\mathbf{H}^{\top}.$$

If $\mathbf{s}^{(j+1)} = \mathbf{0}$, a valid codeword has been found and decoding terminates with $\hat{\mathbf{c}}^{(j)}$. Otherwise, the process continues to the next iteration $j+1$. The magnitude vector $|\mathbf{L}^{\text{ch}}|$ remains fixed throughout all iterations. The iterative process stops when either a valid codeword is obtained or a maximum number of iterations $j_{\max}$ is reached.

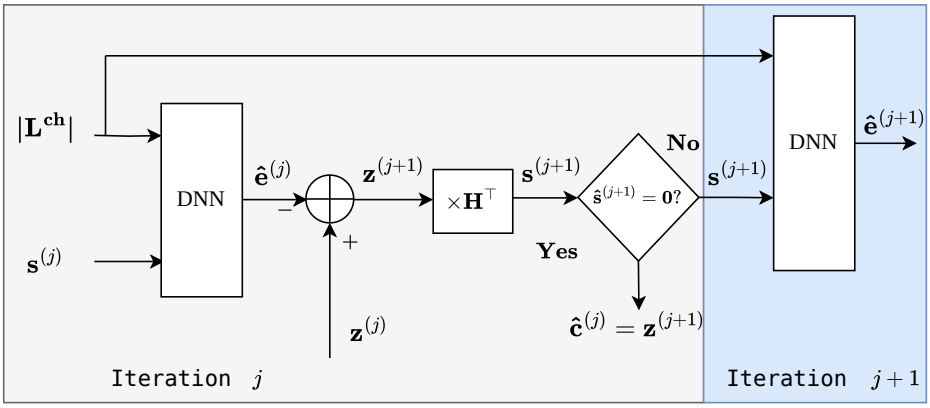


Figure 1.19: A general diagram describing the process of iterative error correction (IEC).

One would think that each prediction incrementally removes noise and improves the likelihood of successful decoding in subsequent iterations. However, our studies suggest a more complex mechanism. Rather than a systematic noise reduction, the model appears to perform a form of random walk over the syndrome space. With each iteration, it encounters a new syndrome, and decoding success is more likely once it reaches a familiar or previously learned syndrome. The overall scheme is illustrated in Fig. 1.19.

Iterative Error Decimation (IED). In contrast to IEC, which refines the entire estimated error pattern across iterations, IED adopts a more conservative correction strategy that flips only one bit at a time based on the decoder's confidence scores.

If the initial prediction (iteration $j = 0$) fails to produce a valid codeword, IED is invoked. The SBND model outputs the soft error probability estimate $\tilde{\mathbf{e}}^{(j)} \in [0,1]^n$ at iteration j. The index of the bit deemed most likely to be erroneous is given by

$$i^{*(j)} = \arg \max_{i \in \{1,\dots,n\}} \tilde{e}_i^{(j)}.$$

We then construct the tentative binary error pattern $\breve{\mathbf{e}}^{(j)} \in \{0,1\}^n$ as

$$\breve{e}_i^{(j)} = \begin{cases} 1, & \text{if } i = i^{*(j)}, \\ 0, & \text{otherwise.} \end{cases}$$

This pattern is used to update the hard-decision vector by:

$$\mathbf{z}^{(j+1)} = \mathbf{z}^{(j)} - \breve{\mathbf{e}}^{(j)}.$$

The codeword estimate becomes $\hat{\mathbf{c}}^{(j)} = \mathbf{z}^{(j+1)}$, and the new syndrome is

$$\mathbf{s}^{(j+1)} = \mathbf{z}^{(j+1)}\mathbf{H}^\top.$$

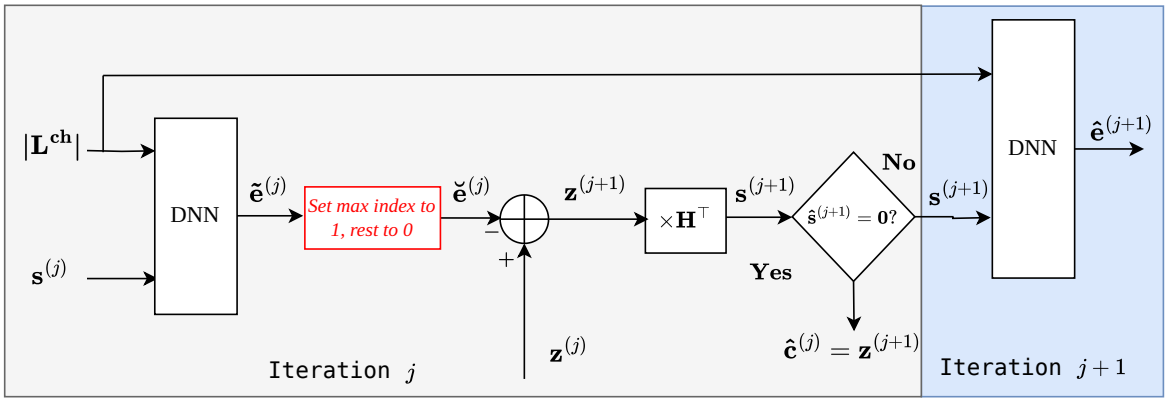


Figure 1.20: A general diagram describing the process of iterative error decimation (IED).

A new decoding iteration is launched using the updated pair $(|\mathbf{L}|, \mathbf{s}^{(j+1)})$ as input to the SBND. This process is repeated iteratively, refining the estimate one bit at a time rather than all at once. The intuition behind IED is that each iteration attempts a more targeted correction addressing the single most suspicious bit. The complete IED scheme is depicted in Fig. 1.20.

Now that both IC strategies have been described, we aim to evaluate their impact on the FER performance of pre-trained SBND models and assess whether one consistently outperforms the other. We consider the GRU(5,5) SBND model trained on the BCH$(63, 45, 7)$ code using 100 million ML error patterns (method 1), as described in previous section. We tested the model with both strategies and the results are presented in Fig. 1.21.

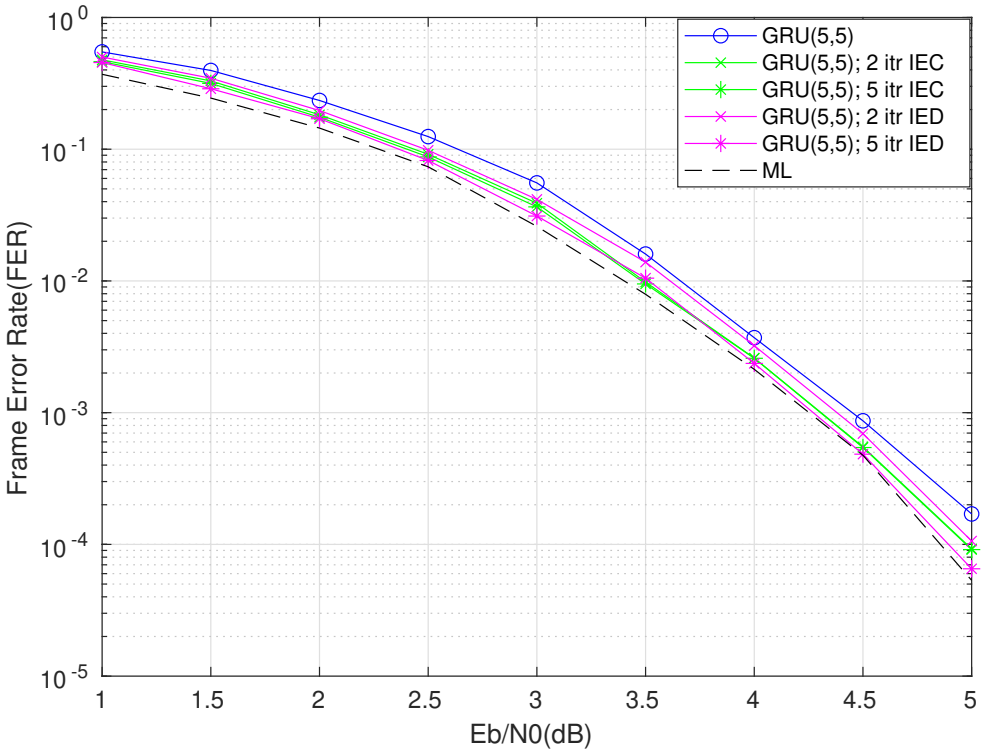


Figure 1.21: Frame error rate for GRU(5,5) model trained on 100M MLD error patterns dataset (method 1) on BCH$(63, 45, 7)$ code with inference time iterative correction.

From the figure, we observe that applying just one additional iteration of IEC significantly reduces FER, bringing performance closer to that of MLD. Further iterations provide only marginal additional improvement, indicating diminishing returns beyond the initial refinement. In comparison, IED initially lags behind IEC at low iteration counts, which is expected since IED targets the correction of a single bit per iteration. This localized correction strategy is less likely to resolve all errors in early iterations. However, with enough iterations, IED can match or even outperform IEC, suggesting that its focused corrections can be effective with enough iterations. Overall, both techniques are effective inference-time enhancements. IEC offers rapid improvement with fewer iterations, while IED shows potential for superior performance when more decoding iterations are permitted.

We then consider the ECCT(64,6) model trained on the BCH$(31, 21, 5)$ code using 4 million ML-IS error patterns (sampling method 3). The results, shown in Fig. 1.22, demonstrate that even transformer-based architectures benefit from iterative enhancement. With just two iterations of IEC, the FER performance already approaches that of MLD. Notably, IED with five iterations achieves a slightly better performance than IEC.

1.7.2 Test-time Augmentation (TTA)

As introduced in Section 1.6.4, code automorphisms can enhance training by introducing input diversity. This principle can also be leveraged at inference time through *Test-Time Augmentation* (TTA), where multiple permuted versions of the received word are decoded in parallel to improve robustness.

An illustrative diagram is given in Fig. 1.23. Let $\mathcal{P} = \{\pi_1, \ldots, \pi_T\} \subseteq \mathrm{Aut}(\mathcal{C})$ be a selected subset of automorphisms of the code $\mathcal{C}$. Let $\mathbf{y}$ denote the received word, and $\mathbf{z}$ its hard decision. For each permutation $\pi_j \in \mathcal{P}$, we construct a transformed input to the model as $(\pi_j(|\mathbf{L}^{\mathrm{ch}}|),\ \pi_j(\mathbf{z})\mathbf{H}^\top)$. The model processes each input and produces a continuous output logits vector $\boldsymbol{\xi}^{(j)}$, which is mapped back to the original bit positions using π_j^{-1}, the inverse permutation. The corresponding estimated binary

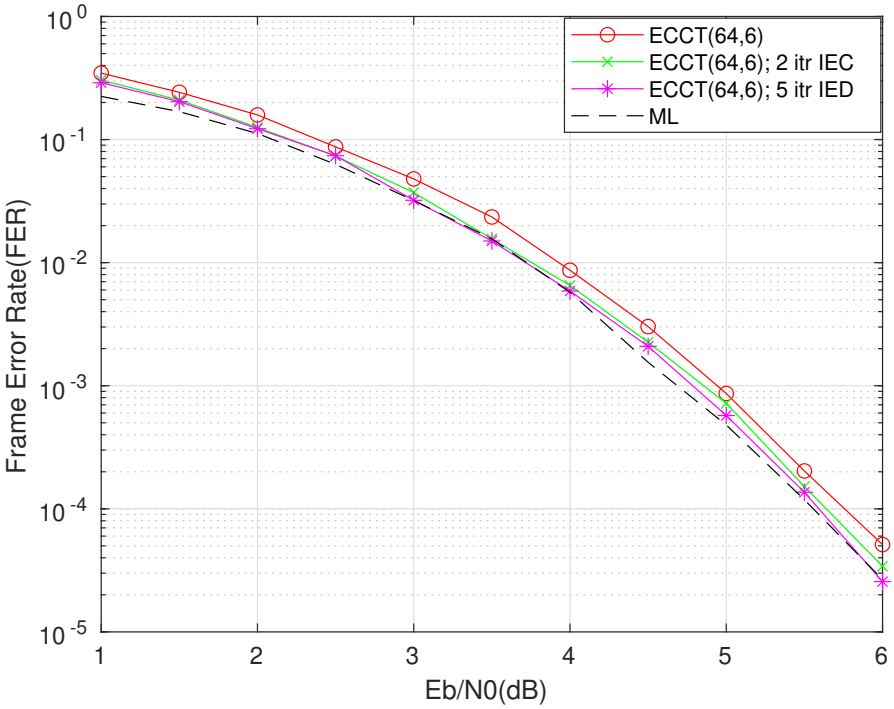


Figure 1.22: Frame error rate for ECCT(64,6) model trained on 4M ML-IS error patterns dataset (method 3) on BCH$(31, 21, 5)$ code with inference time iterative correction.

error pattern $\hat{\mathbf{e}}^{(j)}$ is then recovered as detailed in Section 1.3.2 via:

$$\hat{e}_i^{(j)} = \begin{cases} 1, & \dfrac{1 - \xi^{(j)}_{\pi_j^{-1}(i)}}{2} \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \quad \text{for } i = 1, \ldots, n. \tag{1.28}$$

The decoded candidate codewords are finally given by $\mathbf{c}^{(j)} = \mathbf{z} - \hat{\mathbf{e}}^{(j)}$. Among the T decoded candidates $\{\mathbf{c}^{(j)}\}_{j=1}^{T}$, only those that are valid codewords, $\mathbf{c}^{(j)}\mathbf{H}^{\top} = \mathbf{0}$, are retained in a list $\mathcal{L}$. From this list, a final decision is made by selecting the most likely codeword under the ML criterion, that is selecting the candidate that minimizes the Euclidean distance to the received word $\mathbf{y}$:

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c} \in \mathcal{L}} \|\mathbf{y} - (-1^{\mathbf{c}})\|^2. \tag{1.29}$$

In most cases, there will be no valid codewords in the list $\mathcal{L}$. Therefore, two alternative strategies were explored to aggregate the different model predictions and produce a final decision:

- **Most confident logit selection:** For each bit position $i = 1, \ldots, n$, select the logit with the maximum absolute value across the T candidates:

$$\xi_i^* = \xi^{(j^*)}_{\pi_{j^*}^{-1}(i)} \quad \text{where} \quad j^* = \arg\max_{j=1,\ldots,T} \left| \xi^{(j)}_{\pi_j^{-1}(i)} \right|.$$

- **Averaging candidate logits:** Compute the average logit over all candidates for each bit position:

$$\xi_i^* = \frac{1}{T} \sum_{j=1}^{T} \xi^{(j)}_{\pi_j^{-1}(i)}.$$

After obtaining the aggregated logit vector $\boldsymbol{\xi}^* = (\xi_1^*, \ldots, \xi_n^*)$ by either method (in our case, averaging was found to give better performance), the final estimated error pattern $\hat{\mathbf{e}}$ is recovered by applying

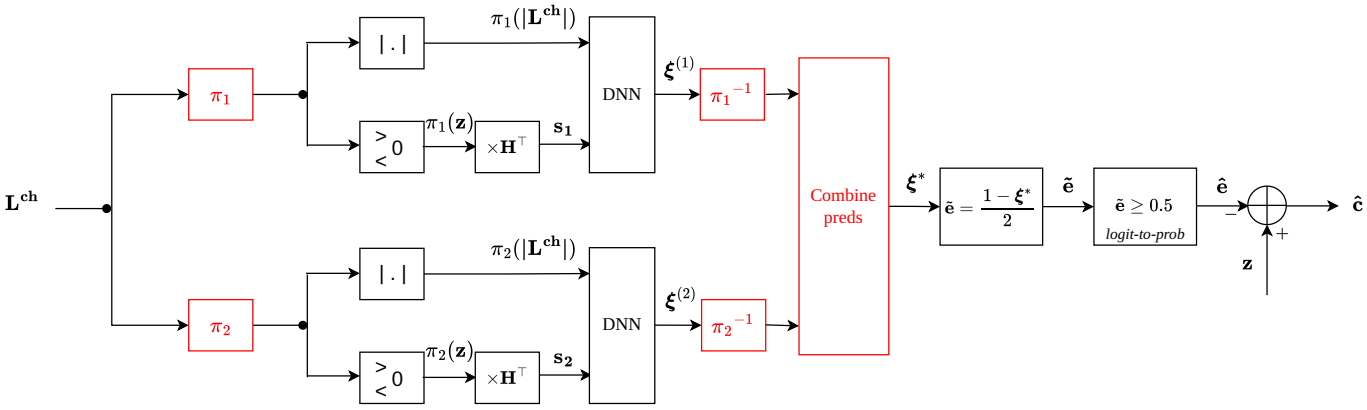


Figure 1.23: An illustrative diagram for the TTA process with $T = 2$ permutations.

the thresholding rule as previously described, and the corresponding decoded codeword is computed as $\hat{\mathbf{c}} = \mathbf{z} - \hat{\mathbf{e}}$.

Our automorphism-based version of TTA for SBND can be related to the *automorphism ensemble decoding (AED)* approach advocated in [23], which exploits the algebraic symmetries inherent in the code's automorphism group to enhance decoding performance. Instead of relying on a single decoding attempt, AED applies multiple decoders in parallel, each operating on a permuted version of the received word generated by different automorphisms. By doing so, it leverages the fact that code automorphisms preserve code structure, allowing diverse yet equivalent perspectives of the same noisy input. This ensemble of decoding attempts increases the chances of successful decoding by effectively combining multiple candidate solutions. In our case, the SBND decoder serves as the base decoder block within the AED framework.

To evaluate the effectiveness of TTA, we consider the GRU(5,3) model trained on the BCH$(31, 21, 5)$ code using the 4M ML-IS dataset (method 3). As shown in Fig. 1.24, TTA provides a noticeable improvement over standard SBND decoding with a single forward pass. In particular, applying TTA with 16 permutations achieves performance comparable to using two iterations of IEC with the same model.

A similar trend is observed in Fig. 1.25 for the GRU(5,5) model trained on the BCH$(63, 45, 7)$ code using the 100 ML error pattern dataset (method 1), where TTA with 16 permutations brings the model's performance significantly closer to that of MLD, while also matching the performance of IEC with five iterations.

1.7.3 Reflection on the inference time SBND enhancement techniques

We have explored some techniques to enhance the performance of already trained SBND models. These approaches are applied at inference time and do not require modifying the model architecture or retraining. IEC and IED refine the decoder's output by iteratively feeding back the model's own predictions, forming a self-boosting loop that helps correct errors missed in a single forward pass. In contrast, TTA adopts an AED-style strategy by leveraging the code's automorphism group to create diverse representations of the received word. Each permuted version is decoded independently, and the outputs are aggregated to improve overall decoding performance. In principle both TTA and self-boosting strategies can be combined but this introduces an additional high computational cost not worthy of the little gain that can be achieved.

That said, both strategies can push SBND models closer to near-MLD FER performance. However, this comes at the cost of increased computational complexity. All such methods require multiple forward passes through the model, which can significantly impact inference time, particularly for larger codes and more complex model architectures. Moreover, it is important to recognize that these enhancements are effective largely because the trained model remains an imperfect approximation

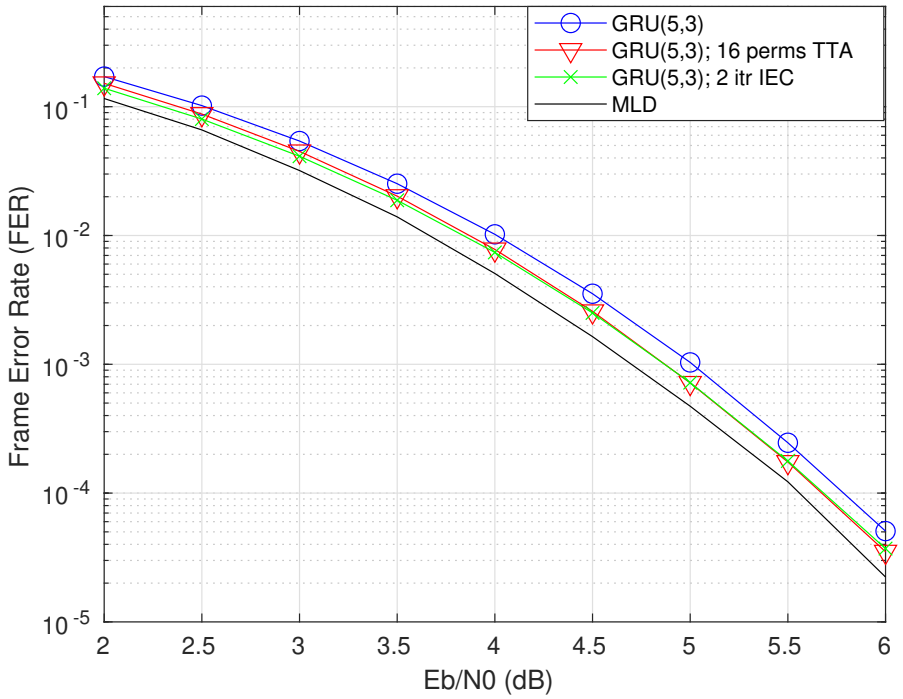


Figure 1.24: FER performance measured with and without data augmentation (TTA) on BCH$(31, 21, 5)$.

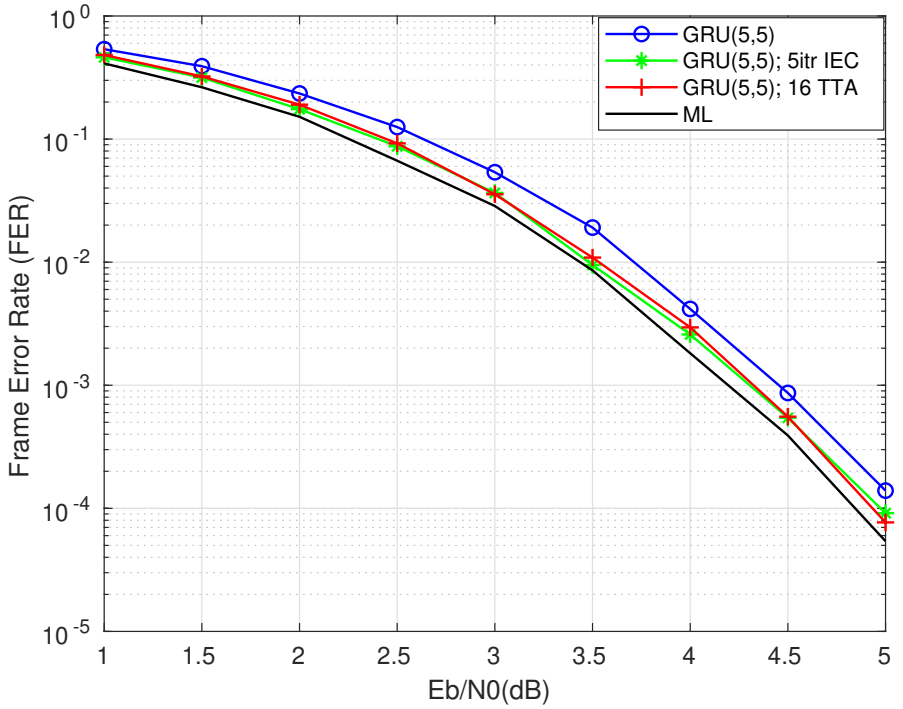


Figure 1.25: FER performance measured with and without test time augmentation (TTA) on BCH$(63, 45, 7)$ compared to IEC.

of MLD. As the model's accuracy improves, the marginal benefits of applying such inference-time techniques diminish.

It is worth noting that if the code of interest is equipped with a hard-decision algebraic decoder (HDD), the most efficient way to improve performance, while keeping inference costs low, is simply to run the HDD on the model's output whenever the prediction is not a valid codeword. Since the models are trained with a BCE loss, which encourages bit-wise error minimization, there is a strong potential for the HDD to correct the remaining residual errors in a significant fraction of cases, as discussed in [24]. This is supported by the results in Fig. 1.26, where the GRU(5,5) model trained on the $t = 3$ error-correcting BCH(63,45,7) combined with a simple HDD achieves performance comparable to other inference-time enhancement techniques especially at higher SNRs.

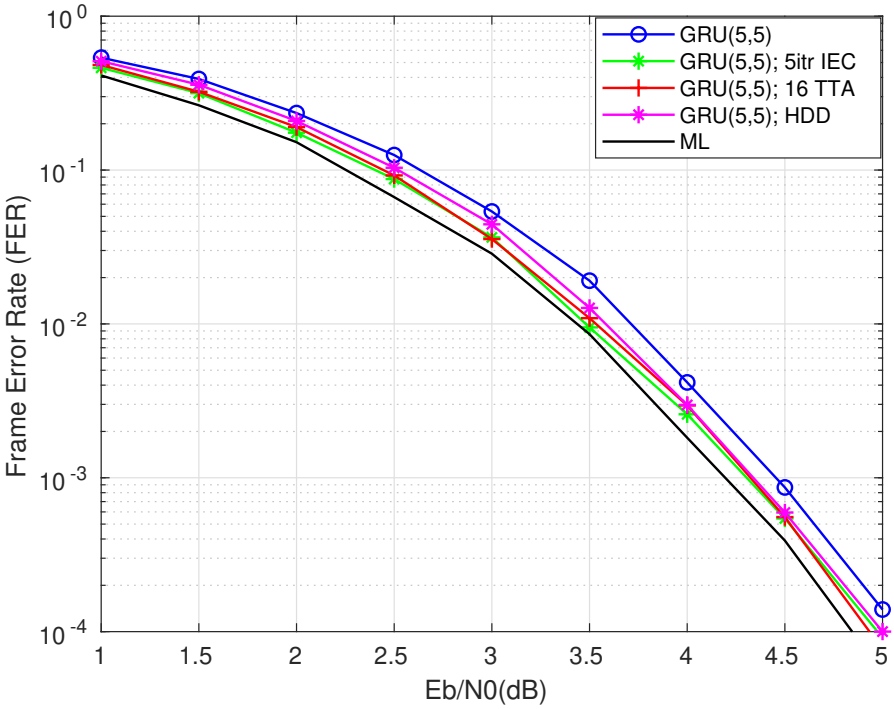


Figure 1.26: FER performance measured with a HDD and compared to inference-time enhancement techniques on BCH(63,45,7).

1.8 Conclusion

In this first contribution to WP3.2 of AI4CODE, we introduced a set of methods aimed at maximizing the performance of SBND models during both training and inference. These methods represent a fusion of best practices from deep learning and classical coding theory.

On the training side, we showed that **carefully curated, fixed datasets** allowed SBND models to **achieve state-of-the-art performance without any changes to the base model architecture**. By **aligning the data with the decoding objective**, incorporating deep learning principles to **optimize the training distribution**, and **introduce diversity through data augmentations**, we enabled more efficient and generalizable learning. To push performance even further at inference time, we explored enhancement techniques inspired by both domains. **Iterative correction** employs the model in a self-improving loop to refine predictions across iterations, inspired by the **boosting technique** of deep learning. From coding theory, we adapted the **Automorphism Ensemble Decoding (AED)** strategy to **Test-Time Augmentation (TTA)**, leveraging the **code automorphism group**. These strategies enable SBND models to **approach near-MLD performance in terms of FER**, **without modifying the model architecture or retraining**, though at the cost of **increased computational complexity**.

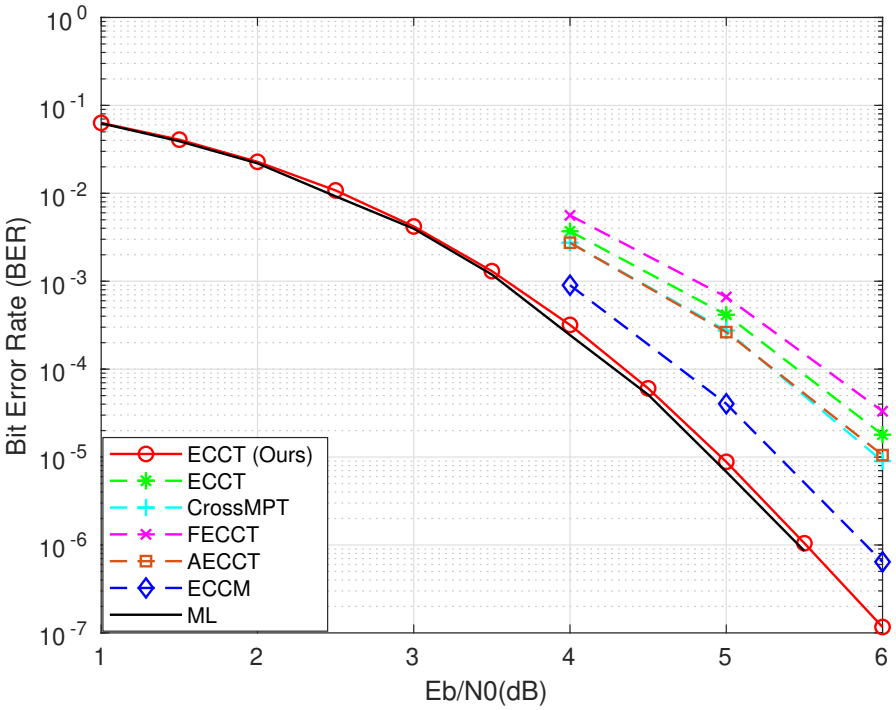


Figure 1.27: BER performance measured of our ECCT(128,6) model trained on BCH(63,45,7) with 64M ML error patterns generated following method 3 and compared to several reference transformer-based SBND models.

In Fig. 1.27, we show the BER performance of our ECCT(128,6) trained on BCH(63,45,7) code with 64M ML error patterns generated according to method 3, comparing it to several reference transformer-based SBND models. These include the original ECCT [2], its improved variant CrossMPT [9], the foundational FECCT model [10], and two recent advancements: AECCT [25], which is an accelerated version of ECCT that optimizes inference speed via sparse attention and early stopping, while retaining the Transformer's global context and ECCM [26] which employs a hybrid Mamba-Transformer decoder leveraging mamba's efficient sequential modeling to further improve inference speed and also introduces a new layer-wise masking strategy to improve performance. All the results presented for these models are taken as is from the respective published papers.

Essentially, all models, except for FECCT, use the **same number of parameters** (1.2M). Our results clearly demonstrate the importance of **data-centric training**: **without modifying the architecture**, our model **outperforms all baselines** simply by **optimizing the training data**. This underscores the power of better data over architectural complexity in neural decoding.

That said, SBND is neither fully accomplished nor one-fits-all solution yet. Consider, for example, the $(96, 48, d_{\min} = 10)$ code from [27], which is a reasonably strong, structured LDPC code. It has 88 VNs with degree 3, 8 VNs with degree 4, 40 CNs with degree 6 and 8 CNs with degree 7. By training a 39M parameter GRU-based model on 64M ML error patterns with data augmentation, we achieve only marginal improvement over BP with 100 iterations. Even with inference-time techniques, the resulting FER remains roughly 1 dB away from the MLD benchmark at an error rate of 10^{-4}, as shown in Fig. 1.28. This underscores a broader point: **while SBND shows significant promise as a model-free approach to achieve near-MLD performance for short block codes, its full potential depends on continued collaboration between deep learning and coding theory**. Coding theory benefits from learning-based generalization, optimization, and flexibility. In turn, **deep learning architectures and training recipes need to be adapted to respect the algebraic structure and constraints of coding problems**. Addressing the remaining limitations of SBND, such as model scalability and architectural efficiency, will require deeper integration of insights from

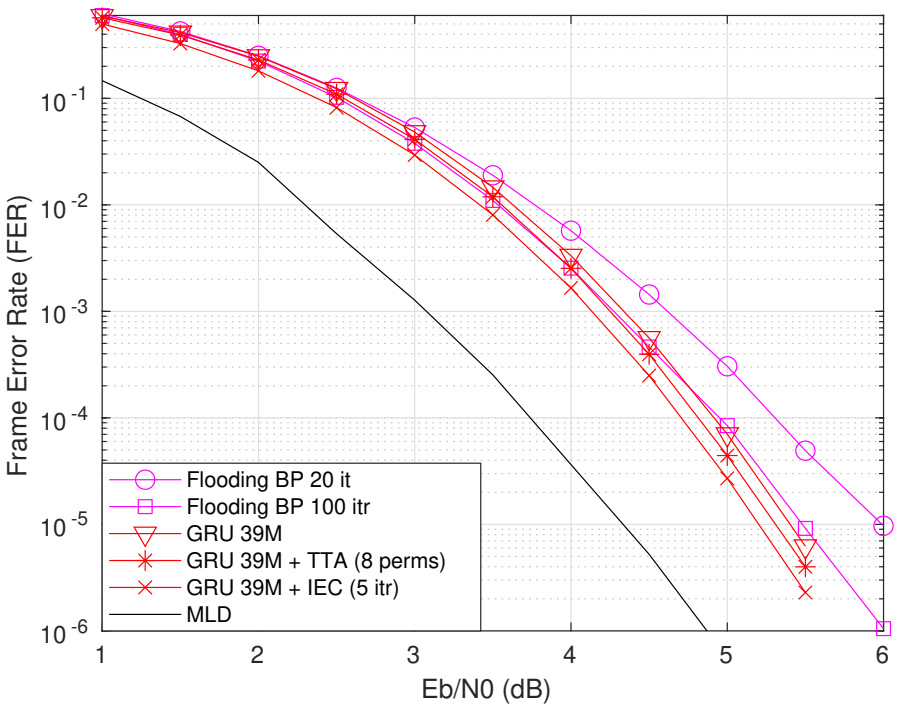


Figure 1.28: FER performance measured with and without inference time enhancements on TUKL$(96, 48, 10)$ and compared to the flooding BP decoder.

both fields. This collaboration is not only fruitful, but necessary to close the remaining gap between neural decoders and optimal decoding.

2 Learning to Improve BP Decoding of Short LDPC Codes

2.1 Motivation

Binary LDPC codes have found numerous applications in digital storage and communication systems, including the latest 5G New Radio. The standard message-passing decoder for LDPC codes is the Belief Propagation (BP) algorithm. Using BP, well-designed long LDPC codes can approach channel capacity within a fraction of decibels, with affordable decoding complexity, only linear in the block length. However, as the block length decreases, BP decoding exhibits an increasing performance gap compared to the optimal Maximum-Likelihood decoder (MLD).

Let us illustrate the problem on a specific code. Consider the $(128, 64, 14)$ rate-1/2 binary LDPC code recommended by the CCSDS for deep-space telecommand links. This quasi-cyclic left-irregular LDPC code is often used for benchmarking short LDPC decoding algorithms in the literature. Its Tanner graph has 64 variables of degree 3 and 64 variables of degree 5. All check nodes have constant degree 8. Its performance under best-effort BP decoding (maximum of 200 iterations) and MLD are plotted in Figure 2.1. For reference purpose, we have also included BP performance with 100 iterations, as well as Polyanskiy's refined normal approximation of the performance achievable by the best possible $(128, 64)$ code [28]. By comparing the MLD performance to this bound, one can see that the short CCSDS LDPC code is not inherently bad, except perhaps for its minimum distance of only 14, which penalizes the high-SNR performance. On the other hand, BP decoding is. For this particular code, the gap to MLD is about 1.5 dB at an FER of 10^{-5}. Clearly, using more iterations does not help. A similar trend is observed with other short LDPC codes of various code rates.

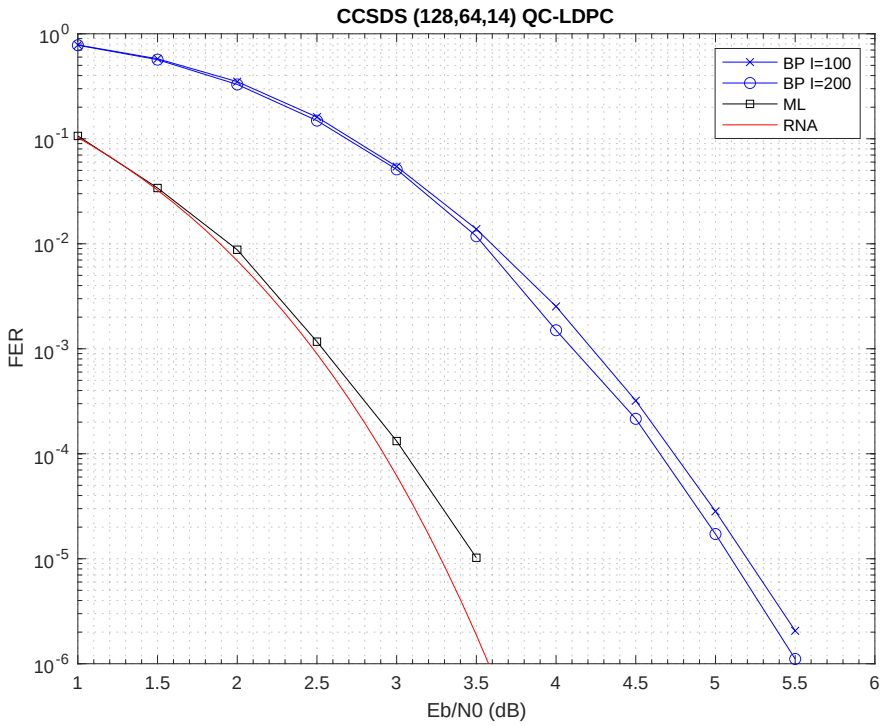


Figure 2.1: Performance of the $(128, 64)$ short LDPC code under BP and ML decoding, and comparison with the refined normal approximation (RNA) on the performance of optimal $(128, 64)$ codes

There exists a variety of techniques for improving the decoding of short LDPC codes. In particular, LDPC codes can take advantage of any near-MLD decoder devised for generic linear codes, albeit not necessarily in the most efficient manner. Ordered-Statistics Decoding (OSD) and its many variants is one of the most powerful approaches among this class. The main problem with OSD is its variable decoding latency with high worst-case complexity, due to a number of re-encoding operations that needs to grow exponentially with the minimum distance when the goal is to match MLD performance. Also, the Gaussian elimination required to construct the most reliable information set for each new received word can prove challenging for the implementation of high-throughput hardware decoders. As a result a hybrid approach is usually preferred, wherein BP decoding is augmented with a low-order OSD post-processor only invoked in case BP decoding fails [29]. The use of learning to improve OSD post-processing will be investigated in Section 3.

We follow a different path in the present study. Driven by the desire to stick with a single decoding algorithm and keep complexity at its lowest, we consider an alternative approach in which some perturbation is applied to the input of the BP decoder whenever it fails, in an attempt to correct some of the channel errors. BP decoding is then restarted for another round. The overall process is repeated several times until a codeword is found or a maximum number of perturbations has been tested. Hereafter, we will adopt the terminology of [30] and refer to this general class of algorithms as *multiple-round BP* (MRBP) decoders. Our interest in this kind of decoders arose from three key observations gathered from experimental evidence:

1. *BP decoding seldom makes errors. On the other hand it fails a lot.* Consider again the simulated BP performance of the $(128, 64)$ CCSDS code shown in Figure 2.1. A minimum of 100 decoding errors were measured at each SNR value. Analysis of the simulation logs reveals that all recorded decoding errors over the whole SNR range were in fact decoding failures. In each case BP failed to converge towards a valid codeword within the prescribed number of iterations. That also means that each time BP found a codeword, it was the transmitted codeword. It may happen that BP converges towards an erroneous codeword, especially at very low SNR. However this remains relatively rare, at least with strong codes, and in such cases, most of the time BP commits an MLD error, *i.e.* an MLD decoder would make the same mistake and decode to the same, incorrect codeword.

2. *When BP fails, most of the time, perturbing a single bit in input is enough to make it succeed.* This observation is best understood with the help of Figure 2.2. This figure was obtained by collecting 10^5 noisy received vectors that made 20 iterations of BP decoding of the $(128, 64)$ CCSDS code fail at an SNR of 4.5 dB. For each of them, we separately flip and saturate[1] the channel LLR of each code bit, leaving the other received bits as is, and then restart another round of BP decoding on the perturbed input. Figure 2.2 shows the empirical distribution of the number of bit perturbations per received vector that led to a decoding success. One can see that for about 75% of the input vectors that cause failure, at least 5 distinct bit flips can lead to a decoding success, the average value being close to 8. It's only for a very few input vectors, less than 1000 out of 10^5, that perturbing a single bit was found not sufficient to recover from the initial decoding failure.

3. *The bits to perturb in order to make BP decoding succeed may not be the ones we could think of at first.* Intuition suggests to prioritise flipping the received bits with smallest channel LLR magnitude, as they are the most-likely to be in error. But experimental evidence reveals that, most often, BP decoding will have no serious difficulty with such obvious errors, unless the received word contains so many of them that the decoder gets overwhelmed. In light of this observation, we have carried out the following experiment with the $(128, 64)$ CCSDS LDPC code. We first run 200 iterations of standard BP decoding. If it fails, we assume a genie decoder

[1]Here and thereafter, by saturating an LLR value, we mean setting its magnitude to ∞ (or to the maximum value allowed by the quantization scheme) and propagating this infinite magnitude in all calculations involving the LLR value, as is done with *frozen* bits in the decoding of polar codes.

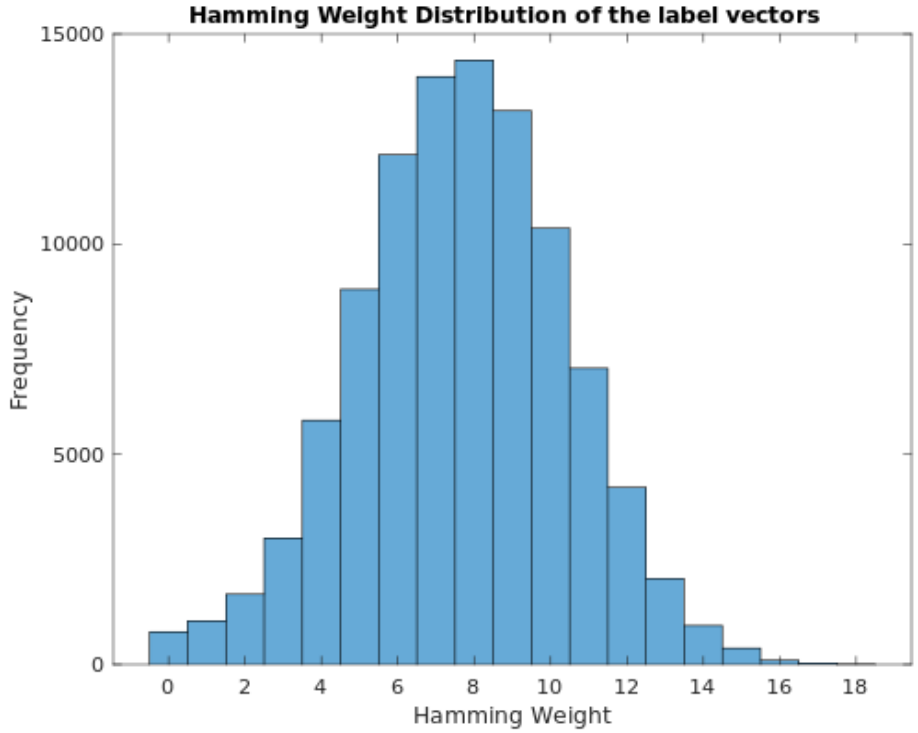


Figure 2.2: Empirical distribution of the number of single bit flips per codeword that can make BP decoding of the (128,64) CCSDS code recover from a decoding failure, at an SNR value of 4.5 dB.

that can tell us exactly which bits are erroneous in the received word. We rank those erroneous bits in order of decreasing channel LLR magnitude, from most-reliable to least-reliable, create F modified received words in which we separately flip and saturate each of the F most-reliable bits in error (MRBE), and then run another round of 200 iterations of BP decoding on each of those F modified received words. This may result in up to F distinct codewords, among which we select the most-likely one as the final decision. The rationale behind this experiment is that we expect the decoder to be more seriously affected by the malicious errors located in the bits it trusts most. The performance of this genie-aided BP decoder is shown in Figure 2.3, for $F = 1$ and $F = 5$ bit flips, respectively. Interestingly, we observe that as few as 5 single-bit flips could be sufficient to reduce the gap to MLD by half, provided the bits to flip are properly selected. Furthermore, these 5 additional BP decoding attempts can be carried out in parallel. Coming closer to MLD performance requires flipping more than one bit at a time.

These observations suggest that, with the proper choice of input perturbations, MRBP could approach MLD in a computationally-efficient manner. In the following, we first describe the principle of MRBP decoding in more details. Then we investigate how to augment MRBP with learning capabilities so as to arrive at a smarter multi-round BP decoder.

2.2 Multi-Round Belief Propagation decoding

2.2.1 General framework

The general framework for MRBP decoding is depicted in Figure 2.4. In the following, we assume the use of an (n, k) binary LDPC code, and denote by $\mathbf{L} = (L_1, \ldots, L_n)$ the channel LLR vector provided at the BP decoder input. For BPSK transmission over AWGN of variance $\sigma_w^2 = N_0/2$, we have $L_i = \frac{2}{\sigma_w^2} r_i$, where r_i is the i-th sample collected at the matched-filter output for some received

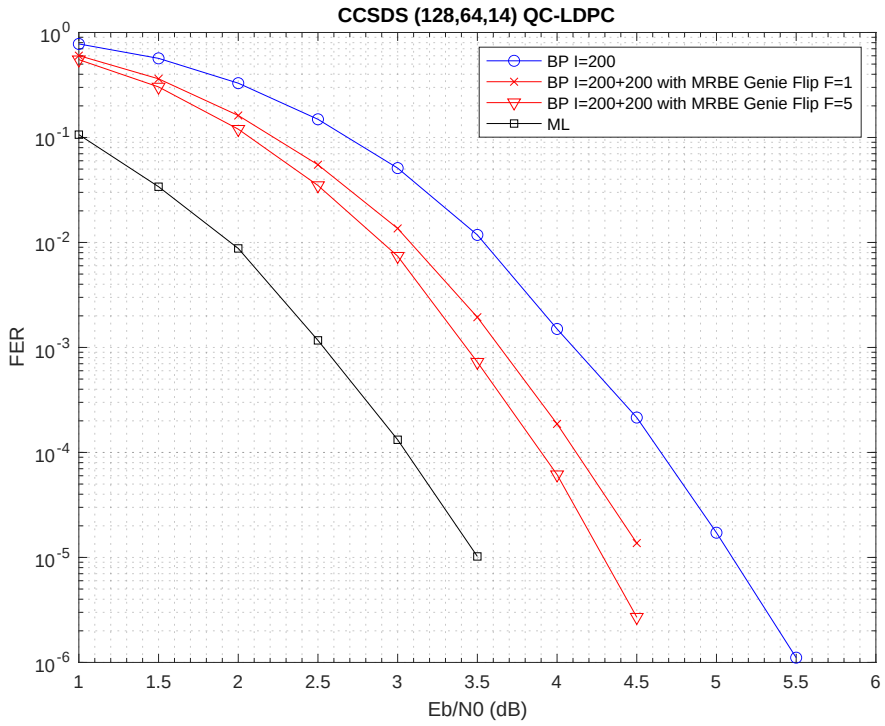


Figure 2.3: Performance of genie-aided BP decoding of the $(128, 64)$ CCSDS LDPC code, where we reprocess the F most-reliable bits received in error (MRBE). By reprocessing, we mean flipping and saturating one of those F bits, leaving all other bits as received, and re-decoding the perturbed vector

word $\mathbf{r}$. An initial round of standard BP decoding is first performed for I_0 iterations, where I_0 is typically in the range $50 - 100$. It is only when this first round fails to decode that we enter the MRBP decoding procedure of Figure 2.4. Detection of BP decoding failure is preferably based on the CRC checksum if payload data includes one, or on the LDPC code syndrome otherwise.

Our description of MRBP closely follows the general description given in [30]. In its most simple form, the first step of MRBP decoding consists in ranking the variable nodes (VN) in the Tanner graph from least reliable to most reliable, according to some custom reliability measure that attempts to identify the bits that are causing a problem to the BP decoder. This ranking is then used to select a subset of low-reliability VNs to perturb. This is the *VN selection step*, and the resulting subset of VNs will be referred to as the *perturbation set*. The perturbation set can be *static* or *dynamic*, depending on whether this set can be computed once and for all from the result of the initial BP decoding attempt, or can evolve with subsequent decoding rounds. A new perturbation pattern is then selected and applied to the input LLR vector $\mathbf{L}$. A perturbation pattern is defined by a set of bits to perturb within the perturbation set, together with an action to apply on those bits. Perturbations can be *deterministic* or *random*. Deterministic perturbations modify the input LLR vector in a deterministic manner. Example of deterministic perturbations include erasing, flipping the sign, and/or saturating the value of selected channel LLRs. Random perturbations consist in introducing some form of random noise into the received word. Example of random perturbation include random sign flips (multiplicative noise) of channel LLRs, or addition of WGN (additive noise). Perturbation of the channel LLR vector $\mathbf{L}$ produces a modified input vector $\mathbf{L}'$ onto which a new BP decoding attempt is performed, using a maximum of $I_1 \leq I_0$ iterations. Typical values for I_1 are in the range $10 - 50$. Here we can either *resume decoding* from the previous failure, using now $\mathbf{L}'$ for the channel LLR values, or reinitialize all messages in the graph and *start decoding anew* on the modified input $\mathbf{L}'$. Restarting the decoding can save memory, whereas resuming decoding allows the VN reliability and set of perturbation patterns to be adapted from one decoding round to the other. If

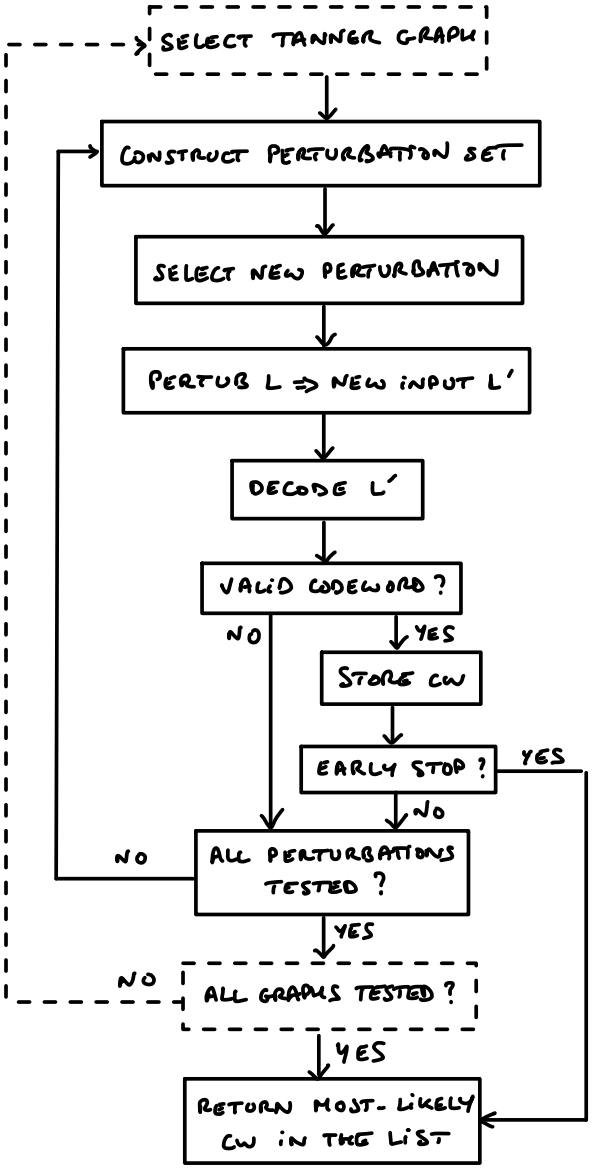


Figure 2.4: Block diagram of a generic multi-round BP decoder

the new decoding attempt fails, the process is repeated with a new perturbation. As for the scheduling of the decoding rounds, we have the choice between testing different perturbation patterns one after the other (*serial scheduling*), or in parallel (*parallel scheduling*). In case decoding succeeds at some round, we can choose to terminate the decoding early and return this codeword. Another option is to store this candidate in a list, and continue with other perturbations, for some total number T of decoding rounds. This may result in up to T distinct candidate codewords, often much less. The decoder decision at the end of the decoding process is then given by the most-likely codeword within this list. Early-stopping on the first codeword can save decoding power, whereas list-decoding most often results in better performance. In case of early-stopping, the ordering of perturbation patterns may need to be optimized for fastest convergence and best performance. Note also that a parallel decoding schedule with a static perturbation set naturally calls for the list-decoding strategy, and minimizes decoding latency. On the other hand, a sequential schedule with early-stop minimizes hardware resources. As depicted in dashed lines in Figure 2.4, the overall process may be repeated on multiple Tanner graph representations of the code, or on different automorphisms (permutations that preserve the code) of the received word, to benefit from decoder diversity.

The first instance of MRBP decoder we are aware of is the *Augmented BP* decoder (ABP) introduced in [31]. Different VN selection rules are proposed and discussed, that all use as primary reliability measure the number of unsatisfied checks a VN is connected to. The perturbation set is formed by the J least reliable bits under the selected reliability measure. It is constructed either in a static or dynamic way, depending on whether decoding is continued or restarted after each perturbation. Both options are discussed and compared in the paper. A total of $T = \sum_{i=1}^{J} 2^j = 2(2^J - 1)$ perturbations pattern is constructed by successively considering all possible combinations of $\pm\infty$ for the channel LLR values within the first j bits of the perturbation set, where $j = 1, ..., J$. The T perturbation patterns are organized in a binary tree of depth J, and different strategies for exploring this tree are discussed (breadth-first vs depth-first), corresponding to the parallel/sequential decoding schedules discussed above. This very comprehensive paper also discusses and compares the two different termination strategies (algorithm A = list decoding vs algorithm B = early-stop on first codeword). The *Saturated Min-Sum* (SMS) decoder of [32] can be regarded as another form of MRBP decoder, if we replace the MS decoder by a BP decoder. Here, the perturbation set consists of the J received bits of smallest channel LLR magnitude, and $T = 2^J$ perturbation patterns are constructed by testing all possible combination of saturated LLR values within the perturbation set. All perturbations are decoded in parallel, with a list-decoding strategy. The SMS decoder is simpler to implement in hardware than the ABP decoder, but has inferior performance for the same total number T of decoding rounds. In [33], an *oscillation-based ABP* (OABP) decoder is proposed that departs from the original ABP decoder in two aspects. First, an improved VN selection rule is used to construct a more relevant perturbation set, that uses the number of sign changes (oscillations) of extrinsic messages from VNs to CNs along iterations as VN reliability measure. This new rule is shown to improve the performance of the ABP decoder, especially with irregular codes, and also make it applicable to punctured codes. Also, the implementation of ABP is made more efficient by smarter exploration and pruning of the perturbation patterns tree. The *Enhanced Quasi-ML* (EQML) decoder proposed later on in [34] advocates nothing more than the same two ideas. More recently, another form of MRBP decoder was introduced in [30]. This *MRBP decoder with impulsive perturbation* (MRBP-IP) differs from ABP and OABP in the VN selection rule, that counts the number of sign mismatches between posterior LLRs and extrinsic messages from VNs to CNs along iterations. This new rule is shown to (slightly) outperform the one used by OABP. But more importantly, MRBP-IP departs from (O)ABP in the way the perturbation patterns are constructed. (O)ABP concentrates its efforts on a relatively small number J of low-reliability bits, and tests all possible combinations of saturated values for the channel LLR values within this subset in an incremental manner. In contrast, MRBP-IP considers an enlarged perturbation set that include the same low-reliability bits as (O)ABP as well as others, of low-to-medium reliability. The kind of perturbation applied, namely impulses, also differs. Here, channel LLRs are either flipped and saturated, or left unchanged, but never saturated while preserving the sign. In other words, we always attempt to correct some of the received bits, and never reinforce their value. In addition, impulses are applied in order of increasing Hamming weight: all single-bit impulses within the perturbation set first, then all combinations of two-bit impulses, etc, until reaching the predefined total number T of perturbation patterns. Lexicographic ordering is applied among impulses of same Hamming weight. In case the perturbation set is formed of the τ least reliable bits for the selected VN reliability measure, and assuming that we test all impulse of Hamming weight $\leq$ some integer $w_{\max}$, then $T = \sum_{w=1}^{w_{\max}} \binom{\tau}{w}$. Compared to (O)ABP, MRBP-IP offers the possibility to flip bits ranked as more reliable for the same total number T of decoding rounds. On the other hand, (O)ABP considers perturbation patterns of larger maximum Hamming weight $w_{\max}$.

As made clear by the previous discussion, MRBP decoders have many parameters, and thus there are as many different MRBP decoders as we can think of. However they all share some common traits. In particular, a comparative analysis of the MRBP decoders proposed in the literature suggests that the performance of this general class of algorithms is primarily governed by

- the VN selection rule and associated VN reliability measure ;
- the kind of perturbation applied (pattern and action) ;

- the total number T of perturbation patterns considered

How to organize the overall decoding, namely the scheduling of the decoding rounds and the choice of the stop criterion, is of utmost relevance for hardware implementation, but appears to be of less importance for the performance. Simulation results for the $(155, 64, 20)$ Tanner code in [31] show a 0.2 dB advantage for the list-decoding strategy over an early-stop on the first codeword, for the same total T of decoding rounds. In light of this observation, *we have chosen to focus exclusively on a list-decoding strategy with a static perturbation set and parallel decoding of the perturbation patterns.* This approach has the additional advantage of having fixed, minimal decoding latency.

In the next sub-section, we summarize the main conclusions drawn from our investigations on how to choose to bits to perturb, and on how to perturb them, for maximal performance. We also give some simulation results to illustrate the typical complexity versus performance tradeoff that can be achieved by MRBP decoders.

2.2.2 How to select the bits to perturb

The VN reliability measure used to rank and select the suspicious VNs that will form the perturbation set clearly plays a central role in the performance of MRBP decoders. Based on our review of the MRBP decoders proposed in the literature, we have identified 7 different reliability metrics that can be used to identify VNs that cause problem to the BP decoder:

1. the channel LLR magnitude
2. the APP magnitude
3. the number of unsatisfied checks the VN is connected to
4. the number of sign changes of the APP along the iterations
5. the number of sign changes of the outgoing extrinsic messages along the iterations
6. the number of sign mismatches between APP and outgoing extrinsic messages along the iterations
7. the number of sign mismatches between incoming and outgoing extrinsic messages along the iterations

The first 6 metrics are taken from the literature, whereas the last one is a contribution of this work. In order to formally define each reliability measure, we need to introduce some notation. Consider some VN of index v in the Tanner graph. Denote by L_v the corresponding channel LLR (input message), and by $\mathcal{M}(v)$ the set of neighboring check nodes (CN) connected to this VN. Let $\mu_{c\to v}^{(i)}$ be the incoming extrinsic message received from some CN $c \in \mathcal{M}(v)$ at iteration i, and $\lambda_v^{(i)}$ be the corresponding log-APP for the VN v, calculated as:

$$\lambda_v^{(i)} = L_v + \sum_{c\in\mathcal{M}(v)} \mu_{c\to v}^{(i)}$$

The outgoing extrinsic message sent back to check node c at iteration i is then given by:

$$\mu_{v\to c}^{(i)} = \lambda_v^{(i)} - \mu_{c\to v}^{(i)}$$

Hereafter we will assume that non-negative (resp. negative) LLRs and messages correspond to a logical 0 (resp. logical 1). Equipped with these notations and conventions, we are now in position to describe how is calculated each of the aforementioned reliability metric.

Channel LLR magnitude (iLLRm)

The channel LLR magnitude for VN v is simply

$$\text{iLLRm}(v) = |L_v|$$

The higher the metric, the more reliable the bit. There is a direct relationship between this metric and the probability that a bit is erroneous at the decoder input. On the other hand, this metric is absolutely not related in any manner to the current decoder state. Also it cannot be used with codes with punctured nodes. This metric does not account for irregularity in the VN degrees. The channel LLR magnitude is the reliability measure used by the SMS decoder [32].

APP magnitude (APPm)

The APP magnitude for VN v is measured at the final iteration I_0, when the initial BP decoding attempt fails, and reads

$$\text{APPm}(v) = |\lambda_v^{(I_0)}|$$

As for the previous metric, the higher the metric, the more reliable the bit, in principle. There is a direct relationship between this metric and the probability that a bit is erroneous at the decoder output. This metric does not account for code irregularity. This is the metric commonly used to select the unreliable VNs to reprocess when combining BP with OSD post-processing [29].

Number of unsatisfied checks (nUC)

This metric lies at the root of all the VN selection rules devised for the original ABP decoder in [31]. It consists in counting, at the final iteration I_0, how many parity-check equations fail among the neighbors of the VN. Formally,

$$\text{nUC}(v) = \sum_{c \in \mathcal{M}(v)} \mathbb{I}\left(\prod_{v' \in \mathcal{N}(c)} \text{sign}(\mu_{v \to c}^{(I_0)}) = -1 \right)$$

where $\mathbb{I}(A)$ is the indicator function for event A, and where $\mathcal{N}(c)$ denote the set of VNs connected to CN c. The lower this metric, the more reliable the bit. This metric tacitly accounts for code irregularity. It is often the case, especially for regular codes, that many VN have the same number of unsatisfied checks. The ABP decoder uses the channel LLR magnitude as auxiliary reliability measure to break ties. In order to break ties based solely on the messages received from the CNs, we chose to rank VNs of equal nUC metric by order of increasing CN reliability, where the CN reliability of a VN is defined as the minimum magnitude among the messages received from its neighbors. Formally, if v and v' are two VNs such that $\text{nUC}(v) = \text{nUC}(v')$, v will be considered as less reliable than v' iff

$$\min_{c \in \mathcal{M}(v)} |\mu_{c \to v}^{(I_0)}| < \min_{c' \in \mathcal{M}(v')} |\mu_{c' \to v'}^{(I_0)}|$$

Number of sign changes on the APP (nSCA)

This metric was introduced in [35] as a way to assess how oscillating a VN can be. Indeed, VNs that change sign frequently are likely to be part of oscillating, periodic or aperiodic trapping sets. The nSCA reliability measure counts the number of times the APP changed sign since the first iteration:

$$\text{nSCA}(v) = \sum_{i=1}^{I_0} \mathbb{I}\left(\text{sign}\left(\lambda_v^{(i)}\right) \neq \text{sign}\left(\lambda_v^{(i-1)}\right) \right)$$

The lower this metric, the more reliable the bit. This metric does not account for irregularity. It can be used with punctured codes.

Number of sign changes on the extrinsic messages (nSCE)

This reliability measure is advocated in [33] as well as in [34] as a replacement to the nSCA metric, that better reflects the confidence to place in a VN and also naturally accounts for its degree, in the case of irregular codes. This metric tracks the oscillations on outgoing extrinsic messages on each edge from one iteration to the other:

$$\mathrm{nSCE}(v) = \sum_{i=1}^{I_0} \sum_{c \in \mathcal{M}(v)} \mathbb{I}\left(\mathrm{sign}\left(\mu_{v\to c}^{(i)}\right) \neq \mathrm{sign}\left(\mu_{v\to c}^{(i-1)}\right)\right)$$

The lower the metric, the more reliable the bit. Tracking the dynamic of sign changes on the extrinsic messages over iterations makes it compatible with punctured nodes. However it cannot be used with codes having degree-1 VNs as the outgoing message is always the same and equal to the channel LLR.

Number of sign mismatches between APP and extrinsic messages (nSMEA)

This last metric was proposed in [30] as an alternative to nSCE for impulse-like perturbations. It consists in tracking the total number of sign mismatches between the APP and all outgoing extrinsic messages over all iterations:

$$\mathrm{nSMEA}(v) = \sum_{i=1}^{I_0} \sum_{c \in \mathcal{M}(v)} \mathbb{I}\left(\mathrm{sign}\left(\mu_{v\to c}^{(i)}\right) \neq \mathrm{sign}\left(\lambda_{v}^{(i)}\right)\right)$$

The lower this metric, the more reliable the bit. This metric takes the VN degree into account.

Number of sign mismatches between incoming and outgoing extrinsic messages (nSME)

We came up with this last metric in an attempt to combine the nSCE and nSMEA reliability measures in a way that only involves the extrinsic messages. It consists in tracking the total number of sign mismatches between the incoming and outgoing extrinsic message on each edge and over all iterations:

$$\mathrm{nSME}(v) = \sum_{i=1}^{I_0} \sum_{c \in \mathcal{M}(v)} \mathbb{I}\left(\mathrm{sign}\left(\mu_{v\to c}^{(i)}\right) \neq \mathrm{sign}\left(\mu_{c\to v}^{(i)}\right)\right)$$

The lower this metric, the more reliable the bit. This metric takes the VN degree into account. It can be used with punctured nodes as well as with codes with degree-1 VNs.

Comparison of reliability measures

Simulation results given for the MRBP decoders found in the literature not only use different VN selection rules, from the above list, but also different forms of perturbation patterns, and different overall decoding logic. In addition, decoders like ABP do not use a single metric but may combine two of them. This makes it difficult to assess whether one VN reliability measure is really better than the others at predicting the bits that hinder the convergence of the BP algorithm. To circumvent the problem and provide a tentative answer to this question, we have evaluated the first 6 candidate metrics in the same simulation conditions. *This benchmark does not include the nSME reliability measure as we came up with it after this study was realized.* The code considered was the $(155, 64, 20)$ LDPC code constructed by R. M. Tanner. This is a $(3, 5)$-regular quasi-cyclic code whose properties and performance are well-known in the literature. The different reliability measures have been evaluated on the same dataset of 50000 noisy received words, generated at a SNR value of 3 dB. The recorded noisy frames were selected so as to make the initial BP decoding attempt fail after $I_0 = 50$ iterations. Upon failure, the 155 bits were ranked according to the selected reliability measure, from least to most reliable. For each reliability measure, we have counted the number of times BP decoding succeeds after flipping and saturating each of the 155 bits, as a function of its rank. $I_1 = 50$ iterations were

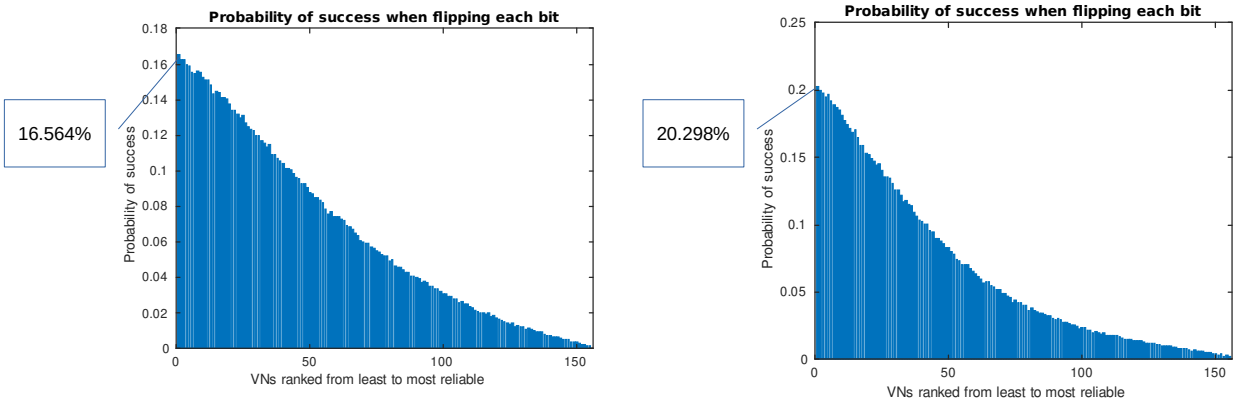
used for the second decoding attempt in all cases. The results are shown in Figure 2.5. In all cases we observe that the probability of decoding success decreases with the rank, as the bit reliability increases. This confirms that all these metrics are relevant at predicting which bits may cause trouble to the BP algorithm. However, not all of them are equally good at this. The oscillation-based nSCE and nSMEA criteria stand out on par with each other, with a probability of correct decoding of 26.5% after a single-bit flip for both. One may legitimately wonder how does the 7th metric (nSME) compare to the nSCE and nSMEA criteria. Simulation results to be presented in Subsection 2.2.4 will provide an answer to this question, and will also illustrate how the probability of decoding success evaluated here can link with FER performance. The iLLRm metric has the worst prediction. This is not surprising as it is solely based on information at the decoder input, that tells nothing about what may have gone wrong later on during the iterative decoding process. This statistical study also reveals that all these VN reliability measures are rather weak when taken individually. Hence the need to test many perturbation patterns in order to approach MLD, as we shall see later in Subsection 2.2.4.

2.2.3 How to perturb the input

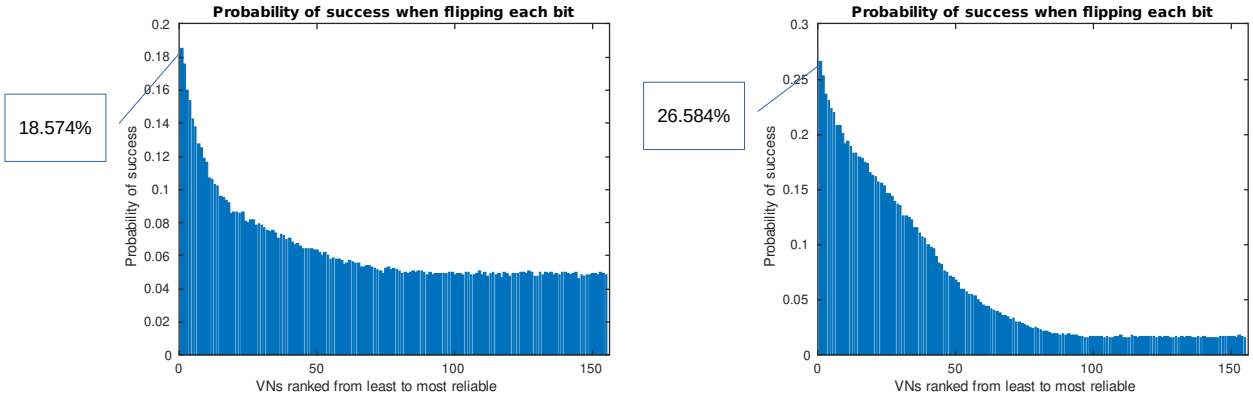
Once the perturbation set is constructed, then comes the question of what kind of perturbation to apply to those low-reliability VNs. As of today, we haven't seen any evidence showing that random perturbations can fundamentally achieve better performance than well-chosen deterministic perturbations. In addition, random perturbation usually exhibit a much larger variability in performance. *Thus, we will only consider deterministic perturbations hereafter.* The main types of deterministic perturbation that can be applied to the decoder soft-input are the following:

- Keep the sign and saturate the channel LLR
- Erase the channel LLR
- Flip the sign of the channel LLR
- Flip the sign and saturate the channel LLR

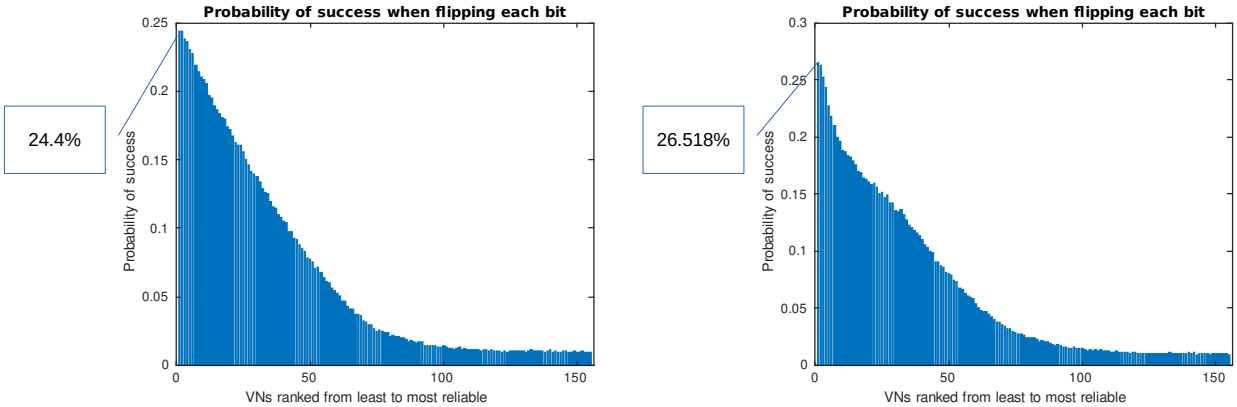
Whereas the first perturbation reinforces the logical value of a received bit, the last two perturbations attempt to correct errors in the received message, either in a soft or hard manner. The erasure option leave it to the decoder to decide what's best for the bit. Since we enter the MRBP decoding procedure after a first decoding failure, it is likely that there is more to gain from attempting to correct errors in the received word than from giving a higher confidence to certain bits. This intuition is supported by the third observation we made in Section 2.1, as well as by the analysis in [31] and [33] which show that it is beneficial to first test the hypothesis of an incorrect sign of the message, before considering the possibility that the sign is correct. Therefore we discard the first perturbation, and have to choose between erasing, flipping, or flipping and saturating the channel LLRs of the bits within the perturbation set. Simulations have been conducted on the $(128, 64)$ CCSDS LDPC code, in order to compare the three options. $I_0 = 100$ iterations were used for the initial BP decoding, and $I_1 = 50$ for the subsequent rounds. VN selection used the nSMEA metric from [30] (see the previous sub-section). The perturbation set was made of the 64 least-reliable bits, and a single-bit perturbation was applied to each bit in the set. The results are depicted in Figure 2.6. They clearly demonstrate that flipping and saturating the channel LLRs is the most effective perturbation among the three.



(a) Left: channel LLR magnitude (iLLRm), right: APP magnitude (APPm)



(b) Left: number of unsatisfied checks (nUC), right: number of sign changes on APP (nSCA)



(c) Left: number of sign changes on the extrinsic (nSCE), right: number of sign mismatches between extrinsic and APP (nSMEA)

Figure 2.5: How good are the various VN reliability measures at predicting which VN to flip to make BP succeed: empirical probability of decoding success after flipping and saturating a single bit, as a function of the rank of the bit, for the $(155, 64)$ Tanner LDPC code at an SNR of 3 dB

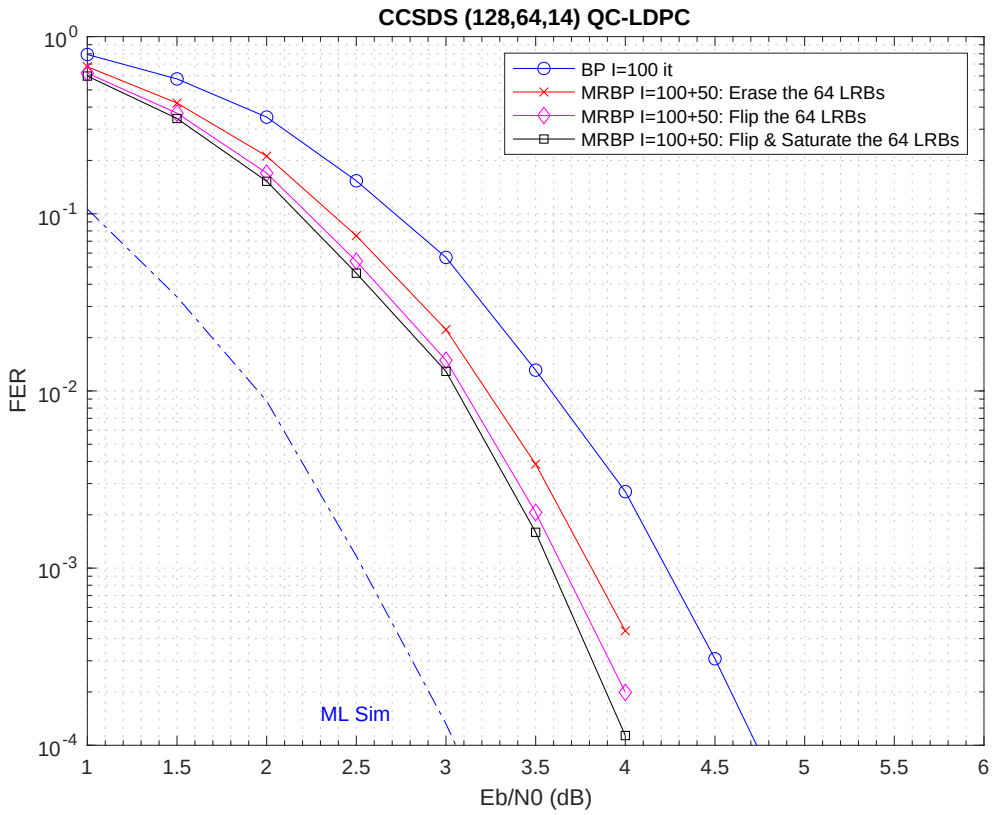


Figure 2.6: Performance comparison between three different kind of perturbations in MRBP decoding of the $(128, 64)$ CCSDS LDPC code based on the nSMEA reliability measure: 1) erase, 2) flip, or 3) flip and saturate each of the 64 least-reliable bits

As for the format of the perturbation patterns, the analysis and simulation results presented in [30] suggest that it might be more advantageous to consider patterns with low Hamming weight but that could involve bits with low to medium reliability, compared to the (O)ABP or SMS strategies that attempt to correct more errors but within a smaller subset of low-reliability bits. For a given total number T of perturbation patterns, the OSD-like generation method of [30] appears to give more freedom in the choice of patterns to test than the Chase-like (O)ABP strategy. Furthermore the third observation in Section 2.1 suggests that approaching MLD may not necessarily require flipping a large number of bits simultaneously. Accordingly we will stick to the method of [30] for building our perturbation patterns in the following. Yet we note that, *to date, there is no clue nor general hindsight about how to balance the size of the perturbation set (maximum rank at which a bit can be flipped) and the maximum hamming weight of the perturbation pattern (maximum number of bits that can be flipped simultaneously) for best performance at fixed total number of perturbation patterns.*

2.2.4 Performance vs complexity

The parameters in MRBP decoding can easily be chosen to meet a wide variety of complexity versus performance tradeoffs. In this subsection, we present simulation results obtained with different codes and decoder configurations in order to highlight the strengths and weaknesses of this class of decoding algorithms.

Let us first consider a regular LDPC code, namely the $(155, 64, 20)$ Tanner code. Figure 2.7 gathers the performance obtained with the following decoders:

- the optimal MLD, which serves as a reference for the other decoders

- a genie-aided two-round BP decoder which flips and saturate the most-reliable bit received in error in case of decoding failure ($T = 1$)
- the OABP decoder of [33] with $J = 6$ decoding layers, $I_0 = 100$ iterations for the first decoding attempt, $I_1 = 10$ iterations for each additional round ; $J = 6$ layers corresponds to $T = 2(2^6 - 1) = 124$ perburbations tested in the worst case, this can be considered as a moderate-complexity decoder
- the ABP decoder of [31] with $J = 11$ decoding layers, $I_0 = 100$ iterations for the first decoding attempt, $I_1 = 10$ iterations for each additional round, and a list-decoding strategy (Alg. A) ; $J = 11$ layers corresponds to $T = 2(2^{11} - 1) = 4092$ perburbations tested in the worst case, this is a best-effort decoder
- the MRBP-IP decoder of [30] with a perturbation set formed of the 64 least-reliable bits for the nSMEA criterion, $T = 64$ single-bit perturbations applied on this set, $I_0 = 100$ iterations for the first decoding attempt, $I_1 = 50$ iterations for each additional round ; this can be considered as a moderate-complexity decoder

One can see that all MRBP decoders achieve notable performance gains compared to the standard BP decoder. Interestingly, the MRBP-IP decoder of [30] is able to match the performance of the more complex OABP decoder with twice as less decoding rounds, owing to the use of impulse-like perturbation patterns that flip less bits but of higher reliability. We have observed a similar trend on other codes. The original ABP decoder with up to 4092 perturbations tested (up to 11 simultaneous bit flips) is only 0.2 dB away from MLD at an FER=10^{-4}. This demonstrates that MRBP decoders are capable of approaching MLD in practice, albeit at the cost of a very large number of perturbation patterns to test. Part of the reason for this might be that the VN reliability measures used so far are not sufficiently accurate at predicting the bits to perturb.

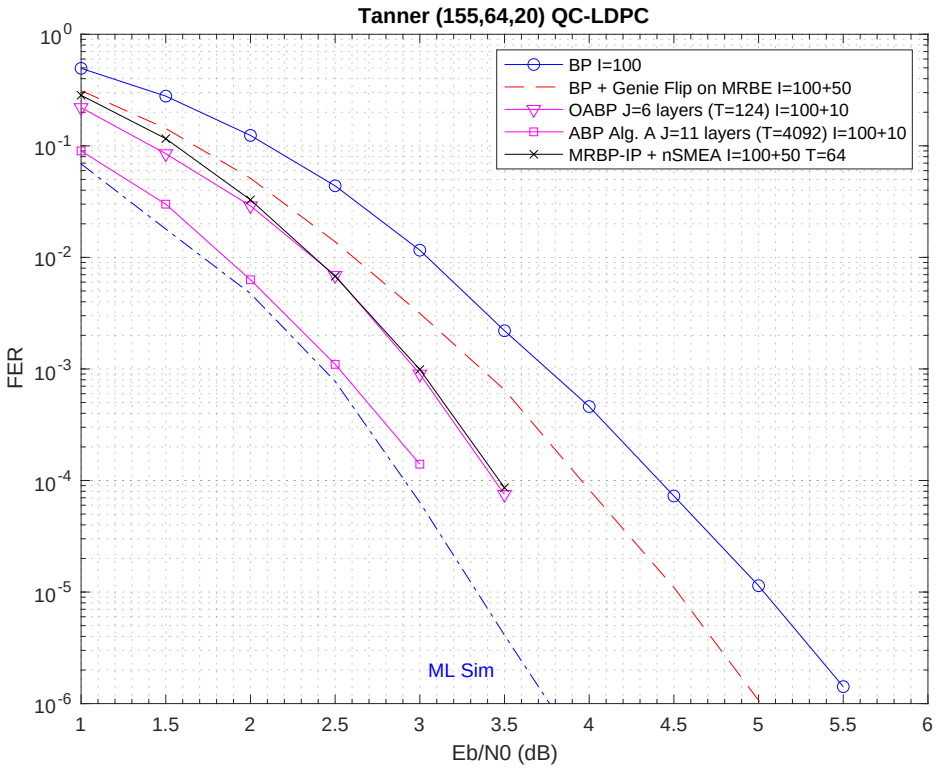


Figure 2.7: Performance of various MRBP decoders for the $(155, 64)$ Tanner LDPC code

Consider next the $(128, 64, 14)$ left-irregular code from the CCSDS TC recommendations. Only MRBP-IP decoding have been investigated for this code, since the study conducted on the Tanner code has shown that MRBP-IP has performance at least as good as (O)ABP decoding. We first compare in Figure 2.8 the performance obtained with the nSCE, nSMEA, and nSME VN reliability measures. The same decoder setting was used for the three simulations: $I_0 = 100$ iterations for the first round, followed by $I_1 = 50$ additional iterations for the next rounds. The perturbation set consisted of the 64 least-reliable bits, and the set of perturbation patterns was composed of all single-bit perturbations within the perturbation set. All perturbed input were decoded in parallel, with the list-decoding strategy. One can see that, on this particular code at least, all three VN reliability measures have essentially the same performance, with perhaps a tiny advantage for the nMSE metric, albeit negligible here. Sticking with the nSMEA metric as VN reliability measure, we then have investigated how the performance evolves with the total number of perturbation patterns. Specifically, we have compared the performance obtained by testing all single bit flips within the 64 least-reliable bits ($w_{\max} = 1$) to the performance obtained by testing all single and double bit flips ($w_{\max} = 2$) within the same perturbation set. In the first case we have a total T of 64 perturbation patterns, against $64 + \binom{64}{2} = 2080$ patterns for the second configuration. The simulation results are plotted in Figure 2.9. Even with 2080 perturbation patterns, the performance is still approximately 0.5 dB away from MLD at an FER of 10^{-4}, suggesting that considerable efforts might be required to approach MLD of this code in this manner.

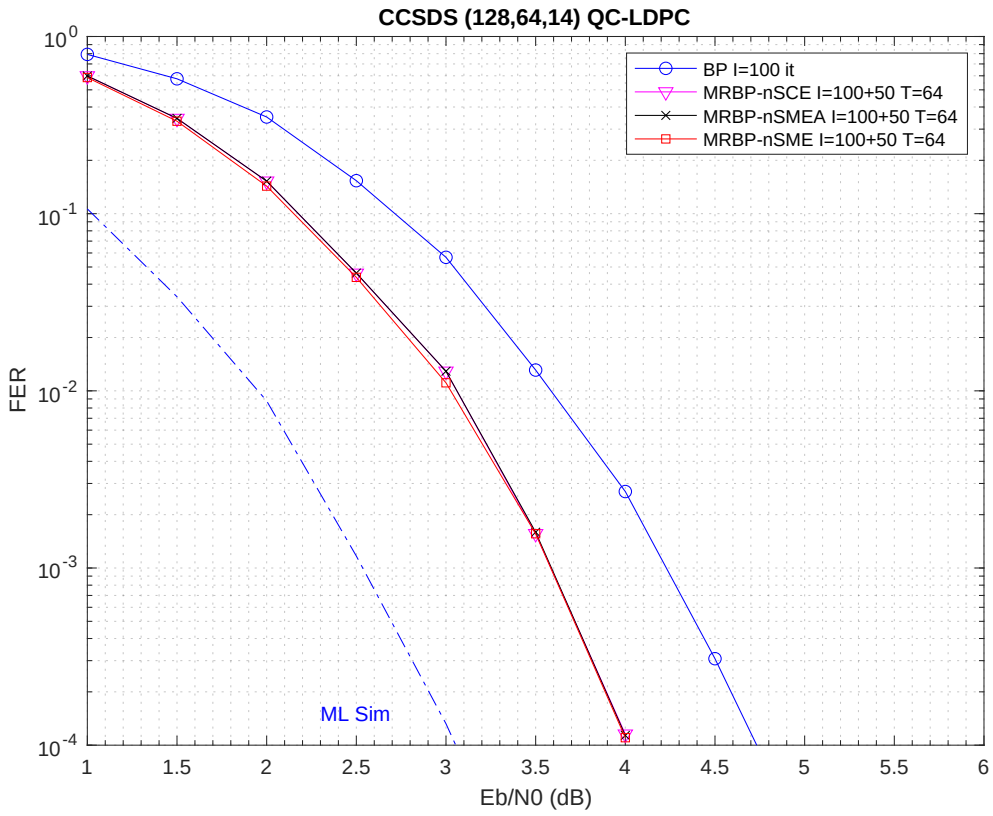


Figure 2.8: Performance comparison between the nSCE, nSMEA and nSME VN reliability measure for MRBP decoding of the $(128, 64)$ CCSDS LDPC code with a total of $T = 64$ single-bit perturbations

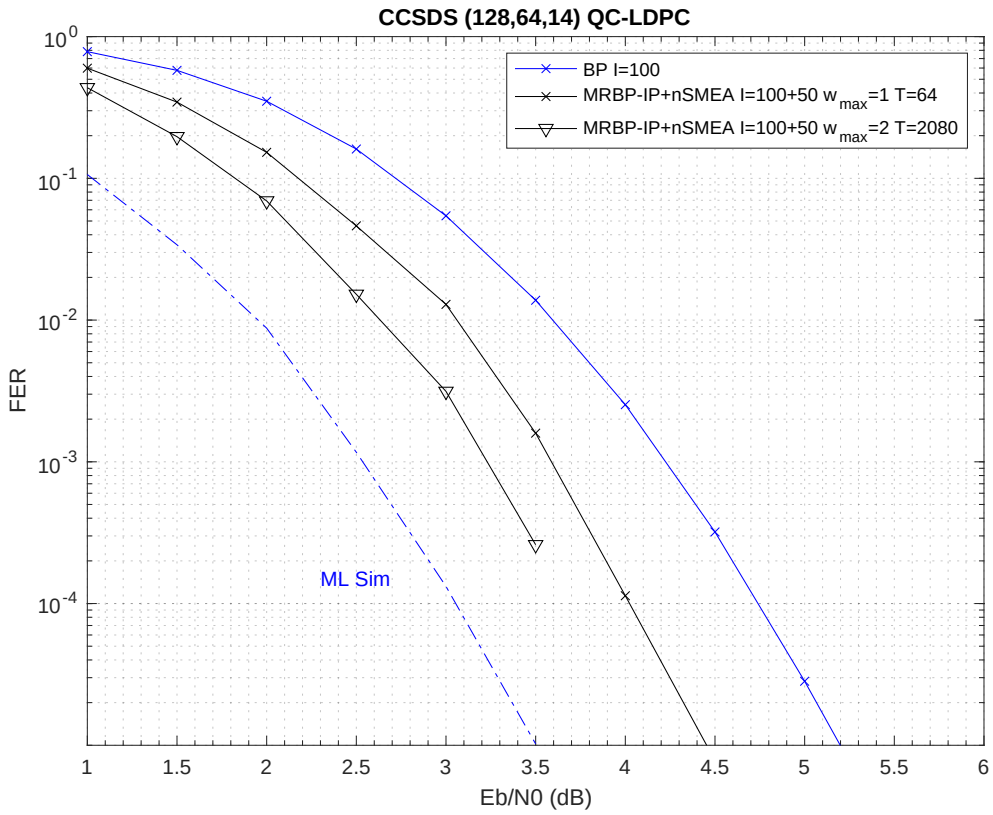


Figure 2.9: Performance of MRBP-IP decoding of the $(128, 64)$ CCSDS LDPC code as a function of the total number T of perturbation patterns, using the nSMEA reliability measure

2.3 Beyond MRBP: Learned MRBP

We have seen that the MRBP decoders proposed in the literature can practically approach the MLD performance of short LDPC codes, but that this may require testing an unreasonably large number of perturbation patterns. We have also shown evidence which suggest that it might be possible to achieve better performance than the MRBP decoders of the literature, or comparable performance but at a much lower computational cost, provided the perturbation patterns are chosen very wisely (ideally pointed out by some genie). Part of the reason for this is that the VN selection rules proposed so far are quite ad-hoc, and prove insufficient capability in guessing correctly which are the bits that hinder the BP algorithm. In this Section, we reflect on how machine learning can be used to arrive at a smarter learned MRBP decoder which would need less perturbation patterns to reach the same performance level than existing expert rules.

To the best of our knowledge, there appears to be little work on this subject in the literature. [36] introduces a syndrome-based decoder which uses reinforcement learning with a proper reward function to guess the received bits that need to be flipped in order to achieve MLD. The proposed algorithm is not specific to LDPC codes, but targets MLD of generic binary linear codes, which is a more much difficult problem in general. All simulation results presented in [36] show at least a 1 dB gap to MLD. More relevant to the present work is [37], which describes a neural-aided MRBP-IP decoder in which a neural network (NN) is used to predict the VN to perturb in place of the expert VN selection rule, directly from the channel LLRs. Only single-bit perturbations are considered. Simulation results show that the proposed scheme can slightly outperform standard MRBP-IP for certain LDPC codes with special structure (the Raptor-like codes of 5G NR). For other codes such as the $(128, 64)$ CCSDS LDPC code, some degradation is observed, leaving much room for improvement. The NN-aided MRBP-IP

approach of [37] has been the baseline on which we have built our investigations.

Two different approaches have been explored and will be outlined in the next two sub-sections. We first apply machine learning techniques in order to learn a better VN selection rule from the ad-hoc metrics found the literature. In a second step, similar to [37], we resort to a neural network to predict directly the perturbation patterns to apply in order to make BP decoding succeed.

2.3.1 Learning a better VN selection rule by combining metrics

Rationale

We have seen that many VN selection rules have been proposed in the literature, and that they all are rather weak at predicting which bit(s) to perturb to make BP recover from a decoding failure. In the experiment of Figure 2.5, the probability of decoding success after perturbing the bit ranked first is about 26% at best. While this observation holds true for each reliability metric taken alone, it is not unreasonable to believe that better prediction could be obtained by combining two or more of these reliability measures. Some of them, for example all the oscillation-based metrics, are obviously highly correlated with each other. There is little hope that combining them will help. But other combinations seem likely to be complementary in nature, as they target different kind of errors. There are essentially three main types of error that will result in a BP decoding failure: convergence to an incorrect codeword (cannot be detected by BP), oscillating trapping sets, and stable trapping sets. An oscillation-based metric is ideally suited to the detection of VNs that are part of oscillating trapping sets, whereas counting the set of unsatisfied checks is expected to be a relevant indicator for detecting stable trapping sets. Both metrics have in common to offer a snapshot of the decoding state at the time of the failure. The input LLR magnitude is another source of reliability information which has the merit of being totally independent of the decoding process. Interestingly, we note that certain decoders, for example the ABP algorithm, already combine two metrics, albeit in a hierarchical manner (one primary reliability measure, assisted by a secondary measure to break ties). Hence the idea of letting the machine learn by itself the combination that works best for our purpose.

Learning setup and experiment

We have applied this idea to MRBP decoding of the $(155, 64)$ Tanner code, with the goal of learning a combination of metrics that could outperform the reference nSMEA metric taken alone. Three different reliability metrics were selected, that we believe to be complementary: the nSMEA metric, the nUC metric, and the iLLRm metric. A bit-level supervised learning approach has been followed, wherein we train a model to predict the likelihood of correct decoding after perturbing a given bit. Bits are processed individually, irrespective of the code structure (codeword framing is not taken explicitly into account in the training). The three reliability metric are regarded as distinct features for the bit of interest. Each metric provides a ranking r for the bit, where r is an integer in the range $[1, 155]$. The smaller the ranking, the less reliable the bit. A dataset was generated from a collection of 50000 noisy words that make BP decoding fail at a channel SNR of 3 dB. This represents a grand total of $N = 155 \times 50000 = 7750000$ bit entries. For each entry, the model input $\mathbf{x}$ is the triplet (r_1, r_2, r_3) of rankings calculated for the bit based on the received word the bit is part of, and on the corresponding decoder state upon failure. The target $\mathbf{y}$ is a binary label equal to 1 if decoding succeeds after flipping and saturating that bit, and 0 if the second decoding round also fails. The binary cross entropy loss function was used to drive the training.

Results

All results reported in this subsection have been obtained with the Scikit-Learn library. The first model we tried was a simple logistic regression, for which we were able to reach a validation accuracy of 75.26%. The probability of decoding success and FER performance of the learned rule are shown in Figure 2.10. In both cases we reach the exact same performance than with the nSMEA VN selection

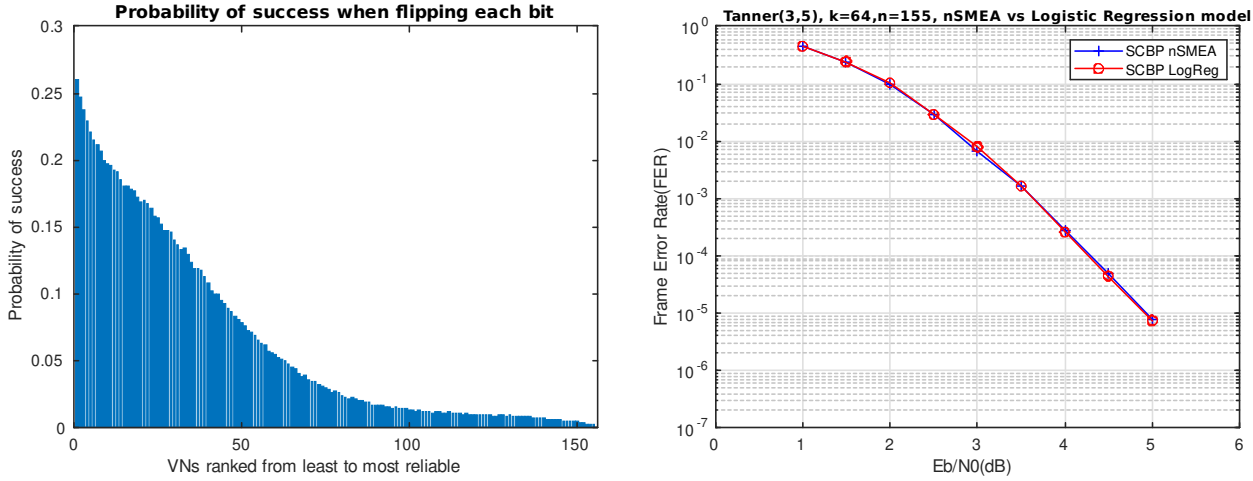


Figure 2.10: Left: Empirical distribution of the probability of decoding success as a function of the bit ranking for the logistic regression model ; Right: Performance comparison between the nSMEA criterion and the VN selection rule learned by logistic regression when applied to MRBP decoding of the $(128, 64)$ CCSDS LDPC code where only the least-reliable bit is perturbed ($T = 1$)

rule alone. The experiment was repeated with other learning models, yielding the following validation accuracy scores:

- 75.45% with random forests
- 75.40% with a multi-layer perceptron having 1 hidden layer only
- 75.38% with a decision tree
- 75.10% using SVM with a linear kernel
- 73.00% using SVM with a polynomial kernel

None of them was able to improve over nSMEA, neither in prediction accuracy nor in practical MRBP decoding performance.

Conclusions

There are at least three possible explanations for this disappointing outcome. The first would be that there is nothing more to learn from the three selected metrics taken all together than from the nSMEA metric alone. However this seems very unlikely given the fact that the iLLRm metric is a form of prior information absolutely not related to the decoding itself. A second explanation could be that it is due to the learning approach we have followed, which discards all information related to the code, including the graphical structure. Indeed we process all code bits individually and on an equal basis, whereas some bits can play a more important role than others in BP decoding. The third explanation could be that the metrics we selected are a too crude summary of the decoder state. The problem can be further aggravated by the choice made of expressing reliability in the normalized form of a ranking, independently on the actual value of the metric. This coarse quantization of metrics could have resulted in an additional loss of information. This is especially the case for metrics like nSMEA where two bits ranked side by side may differ by a large number of sign mismatches, and thus have very different reliability levels, which cannot be captured by the sole rank.

We have decided not to pursue further along this way as it occurred to us that, given that decoding has failed, it might not be prudent to excessively trust metrics solely based on the decoder state upon failure. Rather than relying on some possibly biased expert rule, a better idea could be to let the

machine find by itself which information is relevant (or not), and figure out how to use it for guessing the problematic bits. This is the principle followed by [37] and also the core of the approach outlined in the next subsection.

2.3.2 Learning the perturbation patterns with an MLP

Motivation and goal

Our second approach was motivated by the desire to address the above-mentioned pitfalls, and follows along the same lines as [37]. In [37], a small NN model is trained in a supervised manner to predict the bits to perturb in order to make MRBP-IP decoding succeed. The model input is the decoder input itself (the channel LLR values). The model output is a vector of real numbers in the range $[0, 1]$, that can be interpreted as the chances, as estimated by the model, that flipping a given bit will make decoding succeed. A series of T single-bit flips is then applied on the subset of T bits having the largest likelihood of success at the model output. The NN model consists of an MLP with n inputs, n outputs, and two hidden layers, of size $2n$ and n, respectively. Supervised learning is driven by an MSE loss function with the vector of channel LLRs as model input, and a length-n binary vector as target, in which 1 and 0 denote a success or failure when re-decoding after flipping and saturating the corresponding channel LLR in the input vector, respectively. On the short CCSDS LDPC code, the proposed neural-aided decoder achieves slightly worse performance than a standard MRBP-IP decoder that uses the nSMEA VN selection rule. We attribute this to the fact that, due to the way it is trained, the model could actually be learning nothing more than the iLLRM VN selection rule, or something equivalent. Our comparative analysis reported in Figure 2.5 has shown that iLLRm is slightly worse at guessing which bit to flip than oscillation-based rules like nSMEA. In addition, the channel LLR magnitude carries information that is highly relevant when it comes to decide whether a degree-1 VN should be flipped or not, perhaps even more relevant than counting the number of sign mismatches between the APP of the node and the outgoing extrinsic message (always equal to the channel LLR for a degree-1 VN). That would explain the tiny performance advantage observed with Raptor-like LDPC codes.

Like [37], we want to augment MRBP decoders with a simple MLP trained to predict the bits that need to be perturbed in order to help BP converge towards a valid codeword. Unlike [37], we would like, ideally, to be able to guess the whole perturbation patterns to apply (Hamming weight and composition), instead of working only with single-bit impulses. We chose to approach the problem from two complementary perspectives:

1. On which data do we need to train ?

2. With what kind of model architecture ?

Regarding the training data, it should include both *objective* information (information available at the decoder input, not yet influenced by the later), and *subjective* information that best captures the decoder state upon failure. Objective information can be trusted, subjective information may need to be questioned. The most comprehensive subjective information that we can think of for training would be to keep a record of the whole iterative decoding trajectory, *i.e.* to keep track of all messages exchanged over all edges at each iteration. However this represents a very large amount of data that would make training slow and difficult. We thus have to find some form of sufficient statistics easier to work with. As for the NN architecture, we expect that it should somehow reflect our prior knowledge on the problem, for example the code constraints that tie bits together.

Our first step towards this goal has been to verify that we could design a model that can learn by itself an expert rule, for example the nSMEA VN selection rule. As a second and on-going step, we are currently working on an original learning architecture that combines objective and subjective information in a manner that explicitly account for the graphical structure of the code.

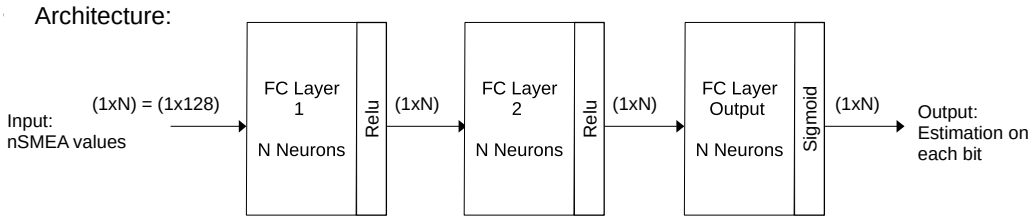


Figure 2.11: NN model architecture used to learn the nSMEA expert rule

Learning an expert rule

We first have designed a NN model trained to estimate the probability of achieving successful decoding after single-bit perturbations, based on a single source of information in input (single feature per bit). The goal of this exercise was twofold: 1) to reproduce the results [37] but with a different VN selection rule, and thereby verify the hypothesis we made on what is really learned by this model, and 2) to make sure we master the basics of NN training with the PyTorch library.

We considered the $(128, 64)$ CCSDS code for this experiment, and a simple fully-connected NN model made of 2 hidden layers, of same size equal to the code length n, followed by an output layer of size n also. As illustrated in Figure 2.11, a ReLU activation function was used at the output of each hidden layer, supplemented by a sigmoid activation function for the final output layer.

To train the model, we collected 10^5 noisy frames that made $I_0 = 20$ iterations of BP decoding fail at a fixed SNR value of 4.5 dB. For each of them, we recorded the length-n integer-valued vector $\mathbf{x}$ of nSMEA values obtained at the last iteration, as well as a length-n binary indicator vector $\mathbf{y}$ where $y_i = 1$ if flipping and saturating the channel LLR of bit i was found to yield a successful outcome after another round of $I_1 = 20$ iterations of BP, and 0 otherwise. The dataset used for supervised learning consisted of the 10^5 $(\mathbf{x}, \mathbf{y})$ pairs, with $\mathbf{x}$ the model input and $\mathbf{y}$ the target. The empirical Hamming weight distribution of the label vectors $\mathbf{y}$ is precisely the one shown in Figure 2.2. In particular, we observe that the average number of ones per received word is approximately 8, with a maximum of 18. Thus we note a strong imbalance between the "0" bit class and the "1" bit class within our dataset. Following standard practice, 67% of the dataset was used for training, while the rest was reserved for testing. The problem of learning the target vector $\mathbf{y}$ is decomposed into n separate binary classification problems, one per code bit. Accordingly, the loss function $\mathcal{L}$ for a particular entry in the dataset was calculated as the sum of the binary cross entropy losses calculated for each bit. Formally, if we denote by $\hat{\mathbf{y}}$ the model output (vector of probabilities) corresponding to the binary label vector $\mathbf{y}$, we have

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

The neural network was trained for 100 epochs with the proposed setup, using the Adam optimizer, with a fixed learning rate of 0.001. The resulting accuracy curves for both training and testing are plotted in Figure 2.12. We observe in particular that the model is able to reach a testing accuracy of about 93.6%. However this value should be interpreted with some caution, as shown by inspection of the confusion matrix which compares the true labels of the test set with the model's prediction:

		Predicted Outcome "0"	Predicted Outcome "1"
True Label	"0"	3929343	42299
	"1"	230639	21719

From the confusion matrix, we can see that model training is affected by the imbalance between 0 and 1 in the dataset. As a result, the model excels at classifying correctly the "0" labels whereas it falls short at classifying the "1" labels, due to uneven exposure during training. Therefore, the trained model ends up predicting most of the time a "0" outcome instead of "1". This automatically translates into a high testing accuracy due to the high number of "0" labels in the dataset, but does not serve us well as what we first and foremost care about for the decoding is to correctly predict the "1" labels. In order to properly evaluate the model for our purpose, we switched to another form of accuracy metric that better relates to our main goal, which is finding what bit flip will most likely lead to a successful decoding. As the model predicts the likelihood of each bit being a "1", we computed on the testing set the probability that the bit with highest likelihood in the model output $\hat{\mathbf{y}}$ indeed corresponds to a true "1" in the testing set. The result was 33.5%. In other words, perturbing the bit with the highest likelihood of decoding success according to the model prediction was found to lead to an actual decoding success at the end of the second round about 33.5% of the time.

The proposed metric better reflects the error-correction performance that we can expect from the trained model than a standard accuracy score. On the other hand, it does not fixes the problem of over-specialization of the model in predicting the "1". Many efforts were placed to account for the dataset imbalance during training. In particular, different cost-adaptive loss functions have been tried, wherein a weight was added to the loss function in an effort to help him pay more attention to "1" than to "0" labels during training. We tried different weighting strategies: per batch, per codeword, and even per bit. However none managed to provide a performance better than what was reported above.

In order to assess how an actual probability of 33.5% of guessing correctly the bit to flip can translate into FER performance, we have compared in Figure 2.13 the performance of MRBP decoding of the $(128, 64)$ CCSDS code where only the least-reliable bit is flipped and saturated ($T = 1$ perturbation), for the expert nSMEA rule and for the rule learned by our model. $I_0 = I_1 = 20$ iterations were used in both cases. One can see from the figure that the FER performance obtained with the

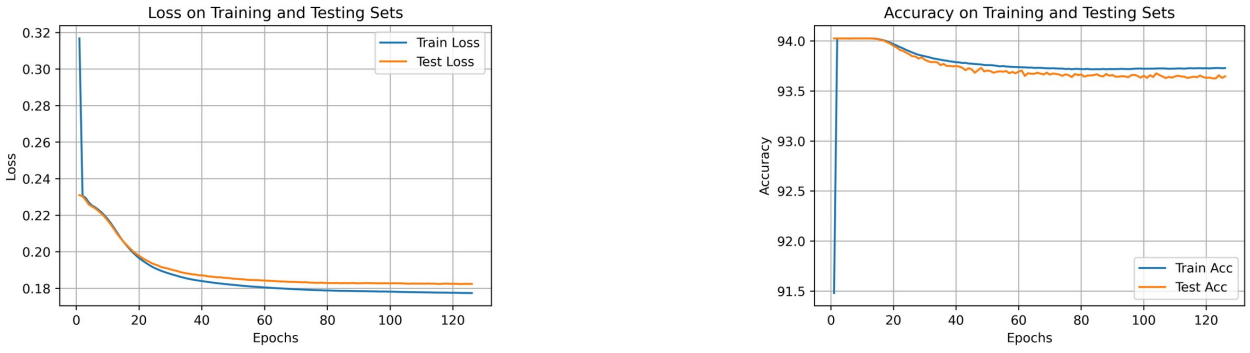


Figure 2.12: Loss and accuracy plots for the NN model trained to learn the nSMEA criterion

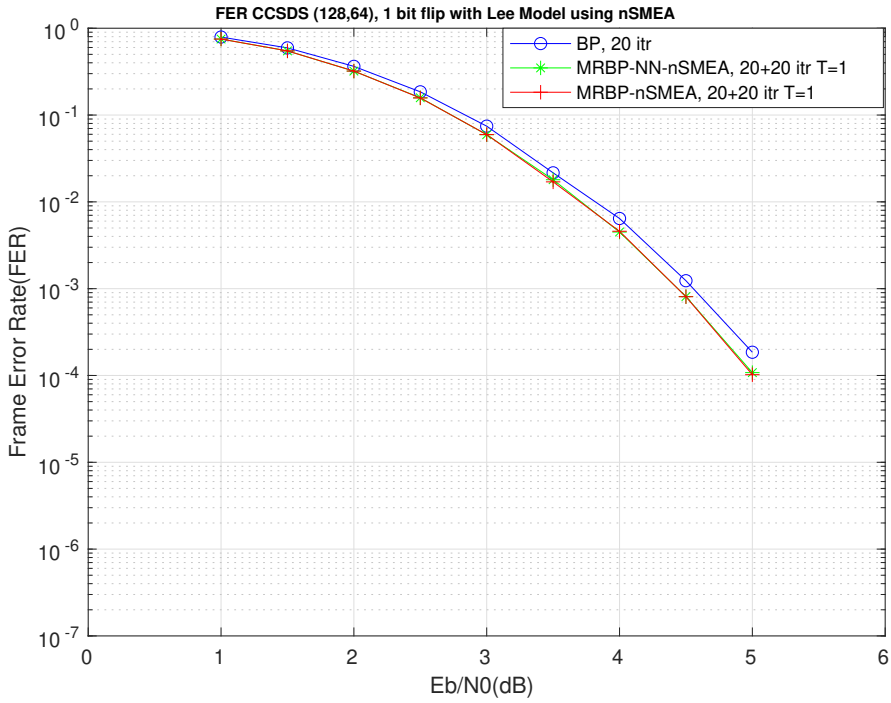


Figure 2.13: Performance of the proposed NN model vs standard nSMEA expert rule for MRBP-IP decoding of the $(128, 64)$ CCSDS code where only the least-reliable bit is flipped

learned rule and with the nSMEA rule completely match. In other words, our trained model managed to match expert's knowledge performance on 1 bit flip. Figure 2.14 shows the results obtained in the same setting but considering now up to T single-bit perturbations within the T least-reliable bits for the selected reliability measure, nSMEA or learned rule. Here again, the trained model perfectly matches the performance of the expert's knowledge. Hence we have been able to design a model that can learn by itself an expert rule, when provided with the proper input.

We note that the nSMEA rule is actually quite simple to learn, so simple that the use of hidden layers is not even necessary in this particular case. Indeed we have verified that the exact same accuracy and FER performance can be obtained with a single-layer MLP having n inputs and n outputs, without hidden layers in-between.

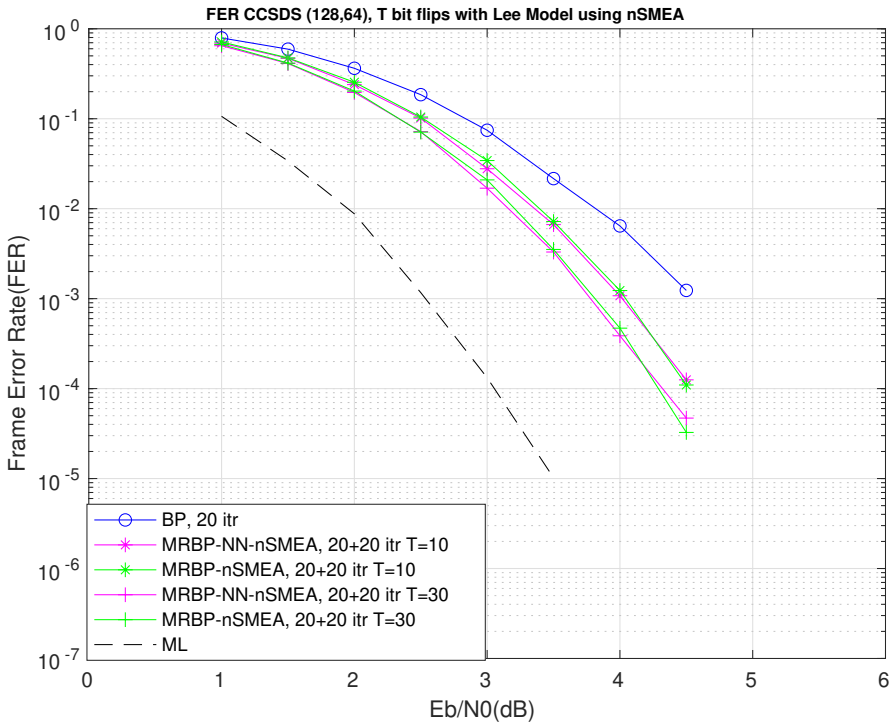


Figure 2.14: Performance of the proposed NN model vs standard nSMEA expert rule for MRBP-IP decoding of the $(128, 64)$ CCSDS code for different number T of single-bit perturbations

As a second, slightly more involved proof of concept, we conducted a similar experiment where the model was trained on a different input, namely the syndrome value observed at the time of the decoding failure. The corresponding NN architecture is shown in Figure 2.15 below. The only difference with the previous model is the size of the input layer, equal to the number $m = n - k$ of syndrome bits (check nodes), 64 in the present case. We used the same supervised learning setup as before, with a different dataset, still composed of 10^5 $(\mathbf{x}, \mathbf{y})$ pairs, but where the model input $\mathbf{x}$ is now the syndrome vector observed at the final iteration. The corresponding loss and accuracy plots are shown in Figure 2.16. Again, the accuracy measured on the test set was about 94%.

Figure 2.17 compares the performance of MRBP decoding of the $(128, 64)$ CCSDS code with $T = 1$ and $T = 10$ single-bit perturbations, using either the rule learned by our model, or an expert nUC rule in which bits are ranked in order of decreasing number of unsatisfied checks they participate in (the higher this number, the less reliable the bit). $I_0 = I_1 = 20$ iterations were used in both cases. It is interesting to note that, once again, the two performance closely match. In other words, the model has been able to infer by itself the expert rule, including the fact that, for each bit, a certain set of syndrome values need to be summed to calculate the final likelihood (the model was able to infer some

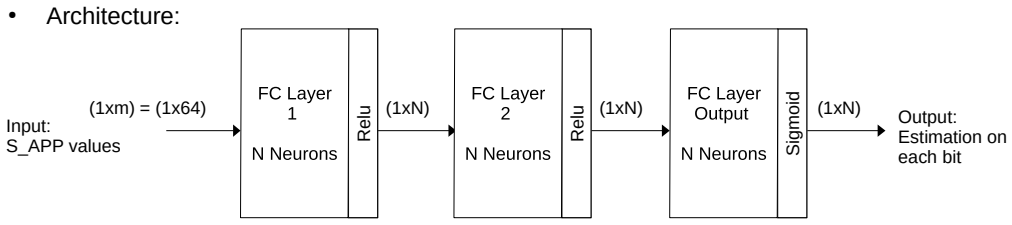


Figure 2.15: NN model architecture used to learn the nUC expert rule

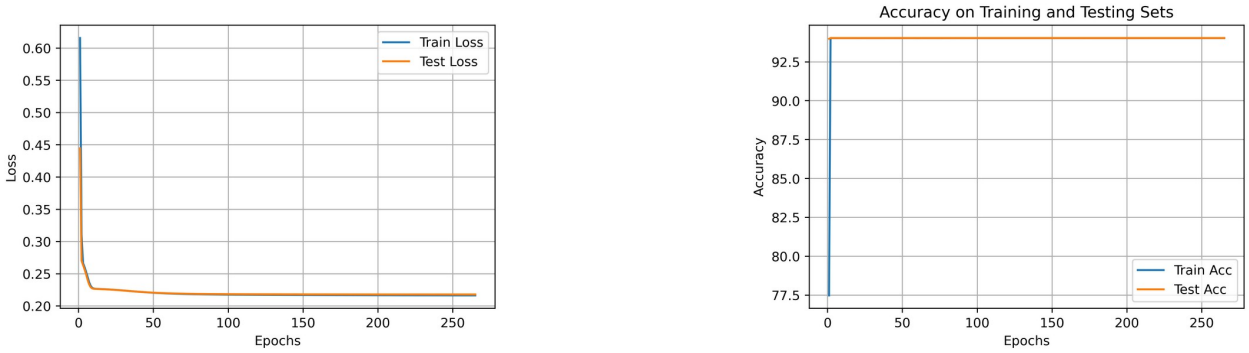


Figure 2.16: Loss and accuracy plots for the NN model trained to learn the nSMEA criterion

of the code constraints).

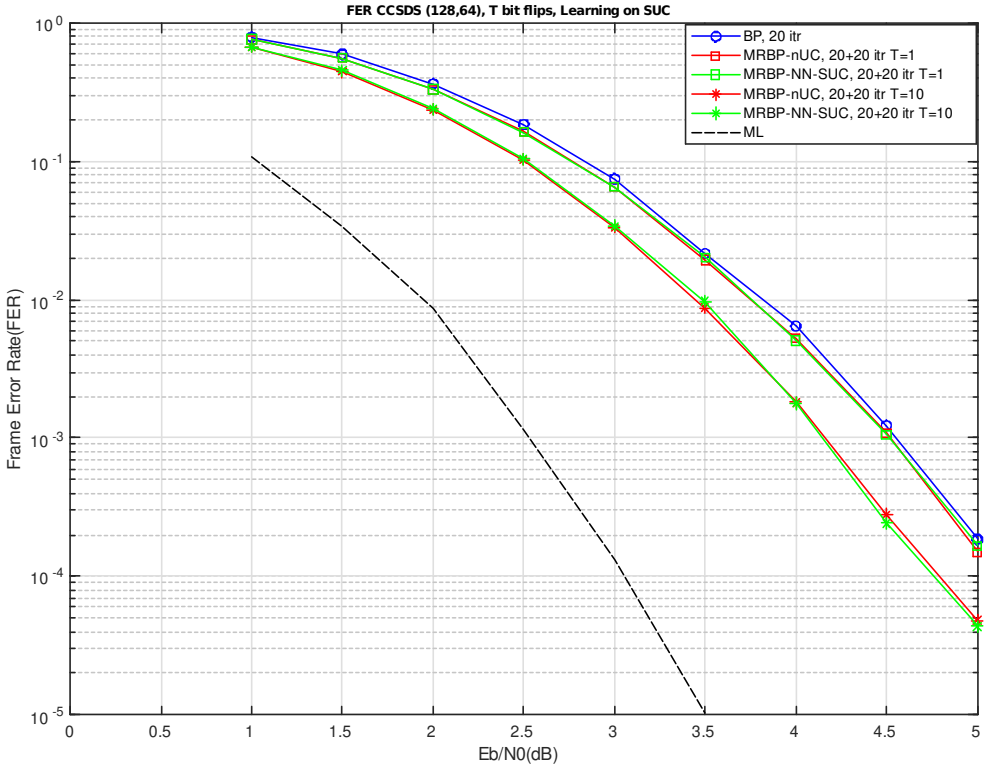


Figure 2.17: Performance of the proposed NN model vs expert nUC rule for MRBP-IP decoding of the $(128, 64)$ CCSDS code for different number T of single-bit perturbations

Learning more than a single expert rule

We tested multiple combinations of expert rules as input to Lee's architecture [37], but none outperformed the nSMEA expert criterion. This led us to fundamentally question and rethink the architecture used to predict the bits to be switched, as explained in the next sub-section.

2.4 Deep learned MRBP

In the previous sections, we introduced MRBP decoding as a powerful method to improve BP decoding for short LDPC codes, enabling it to get closer to MLD performance. However, MRBP requires an unreasonably high number of decoding attempts to reach this performance level. MRBP operates by identifying, whenever BP decoding fails, a set of problematic variable nodes (VNs) that hinder BP convergence. Perturbations are then applied to these VNs, followed by subsequent decoding attempts. We demonstrated that the expert-based rules used to select the VNs to perturb are rather weak, motivating the need for improvement. To address this, we turned to learning-based methods. Our first idea of learning a combination of expert rules bitwise and by standard machine learning techniques was unsuccessful. We then turned our attention to the more capable shallow MLP model from [37], which adopts a codeword-level learning approach where the learning algorithm leverages information related to the entire codeword to decide which VNs to perturb. Unfortunately, here also, the NN was not able to surpass the best-performing expert rule in identifying the most critical VNs to perturb. This shortfall may stem from several factors, motivating for further investigations. In what follows, we propose some modifications on the architecture of the NN model and its input that enhance the model's ability to identify the most critical VNs for perturbation. The central contribution lies in demonstrating that by leveraging upon the principles and models of SBND investigated in Chapter 1, we can predict more accurately which VNs should be perturbed, thereby outperforming expert rules.

The work reported in this section has been published in:

[ILDA25c] A. Ismail, R. Le Bidan, E. Dupraz, and C. Abdel Nour, "Learning Variable Node Selection for Improved Multi-Round Belief Propagation Decoding" in Proc. of *International Symposium on Topics in Coding (ISTC)*, Los Angeles, CA, August 2025.

The following sections and results are an excerpt of Ahmad Ismail's PhD manuscript. We refer the interested reader to this document for complementary details and explanations.

2.4.1 Leveraging SBND to improve MRBP

This sub-section presents the core contribution of this chapter: inspired by SBND, we propose modifications to the NN-aided impulsive perturbation decoding method [37]. These modifications are based on the observation that the problematic VNs that prevent the BP decoder from converging are a subset of the full set of channel-induced errors. This suggests that the task of predicting which VNs should be perturbed during impulsive decoding can be regarded as a relaxed version of the full channel error estimation problem. SBND, as originally proposed, has demonstrated reasonable effectiveness in estimating the complete channel error pattern. Therefore, we see the potential to leverage the principles and models developed for SBND to improve the performance of NN-aided impulsive perturbation decoding. In the following, we outline the key differences between the proposed architecture and the original NN-aided MRBP from [37].

Revisiting the model architecture

Since the NN-aided MRBP approach in [37] uses a lightweight MLP to predict which VNs to perturb, the model's performance may be limited. In addition to that, in the context of SBND, [1] reported that the stacked GRU model is more capable in inferring the full channel errors. To this end, we propose replacing the original MLP in the NN-aided MRBP decoder with a stacked-GRU as implemented in section 1 to potentially enhance its ability to identify the problematic VNs for perturbation.

Revisiting the model input and training setup

From equation (1.18) in section 1, we have established that the pair $(|\mathbf{L}^{\text{ch}}|, \mathbf{s})$ constitutes a sufficient statistics for MLD. Given the proven effectiveness of this input representation in the SBND framework, we adopt it for our NN architecture. While both this input representation and the signed channel

reliabilities used in [37], should capture the same information, the inclusion of the syndrome could potentially provide a clearer picture of the underlying code structure. Furthermore, in this new input representation we normalize the channel LLR magnitude following the min-max normalization rule

$$\tilde{\mathbf{y}} = \frac{\mathbf{y} - y_{\min}}{y_{\max} - y_{\min}}, \tag{2.1}$$

where $y_{\min} = \min_i y_i$ and $y_{\max} = \max_i y_i$, to obtain a normalized version denoted by $\widetilde{|\mathbf{L}^{\text{ch}}|}$.

As seen in shapter 1, an important advantage of $(|\mathbf{L}^{\text{ch}}|, \mathbf{s})$ is that it eliminates the dependence on the transmitted codeword, allowing us to consider the all-zero codeword without loss of generality.

The NN is trained in a supervised manner using a labeled dataset composed of input–output pairs at the codeword level. Randomly generated codewords are transmitted over a BI-AWGN channel at a range of SNR values, followed by BP decoding with I_0 iterations. The N_{tr} received words that result in BP decoding failures are retained for further processing. For each BP decoding failure and each vn_i, where $i \in \{1, \ldots, n\}$, a single-bit perturbation is applied to the channel LLR L_i^{ch}, and BP decoding is re-executed with I_1 iterations. This procedure yields a binary target vector

$$\mathbf{b} = (b_1, \ldots, b_n),$$

where $b_i = 1$ if perturbing vn_i leads to successful BP decoding in the second attempt, and $b_i = 0$ otherwise. For each BP failure instance, we obtain one input–output pair. The complete training dataset is therefore composed of N_{tr} pairs:

$$\mathcal{D}_2 = \left\{ \left(\widetilde{|\mathbf{L}^{\text{ch}}|}, \mathbf{s}\right)_j, \mathbf{b}_j \right\}, \quad j \in \{1, \cdots, N_{tr}\}, \tag{2.2}$$

Compared to the MLP model in [37], the drawback of incorporating the syndrome into the input is that the model's input dimension now scales with the sum of the code length and the number of parity bits.

As in SBND, we formulate the task of identifying the VNs to perturb as a binary classification problem in which the model predicts, for each VN, whether it should be perturbed to enable successful decoding. Accordingly, we train the network using the standard binary cross-entropy (BCE) loss, between the probability estimates $\hat{\mathbf{b}}$ and the ground-truth label vector $\mathbf{b}$, instead of the weighted MSE adopted in [37]:

$$\mathcal{L}_{\text{BCE}}(\mathbf{b}, \hat{\mathbf{b}}) = -\frac{1}{n} \sum_{i=1}^{n} \Big(b_i \log \hat{b}_i + (1 - b_i) \log(1 - \hat{b}_i) \Big). \tag{2.3}$$

Finally, we observed that training the models on a single SNR point was sufficient to achieve good generalization over a range of SNRs; a trend also observed in SBND.

2.4.2 Performance of the proposed deep learned MRBP decoder

We aim to compare the performance of the proposed SBND-inspired learning architecture to the original learning architecture of [37]. To this end, we consider the TUKL(96,48) short LDPC code. We use $I_0 = 20$ decoding iterations for the first BP decoding attempt, and $I_1 = 20$ also for subsequent ones.

Baseline

Before analyzing the specific contributions of each major modification we introduced (stacked GRU architecture, different input representation), we need to establish a baseline reference. Unlike [37], our approach formulates the task as a binary classification problem, trains the model using the BCE loss, and uses training data collected at a single SNR point. We first need to verify that these changes alone do not negatively affect the training process.

To this end, we train the MLP described in [37] on a first dataset $\mathcal{D}_1 = \{(\tilde{\mathbf{y}}_j, \mathbf{b}_j)\}, \quad j \in \{1, \ldots, N_{tr}\}$ of 1M BP decoding failure examples, with normalized signed channel LLRs in input, and with two key differences with respect to [37]: the weighted MSE loss function is replaced with the BCE defined in equation (2.3) (accordingly a sigmoid activation function is applied on the model's output), and all BP failure samples are collected at a single SNR of 3 dB. This baseline model will allow us to isolate the effects of the proposed modifications in subsequent experiments. The results are shown in Fig. 2.18. We can see that these small changes not only did not affect the performance of the model but they even allowed the model to perform better. Now the NN-aided MRBP denoted by MRBP-NN-BCE matches the performance of the MRBP with the iLLRm criteria for $T = 5$ decoding attempts. Further studies showed that training on a single SNR point (3 dB) while keeping the WMSE loss function led to a noticeable improvement over the MRBP-NN-WMSE model trained on samples collected across 1–4 dB. In contrast, replacing WMSE with BCE alone yielded little to no gain. However, when both modifications were applied together, they resulted in the more pronounced improvement reported here.

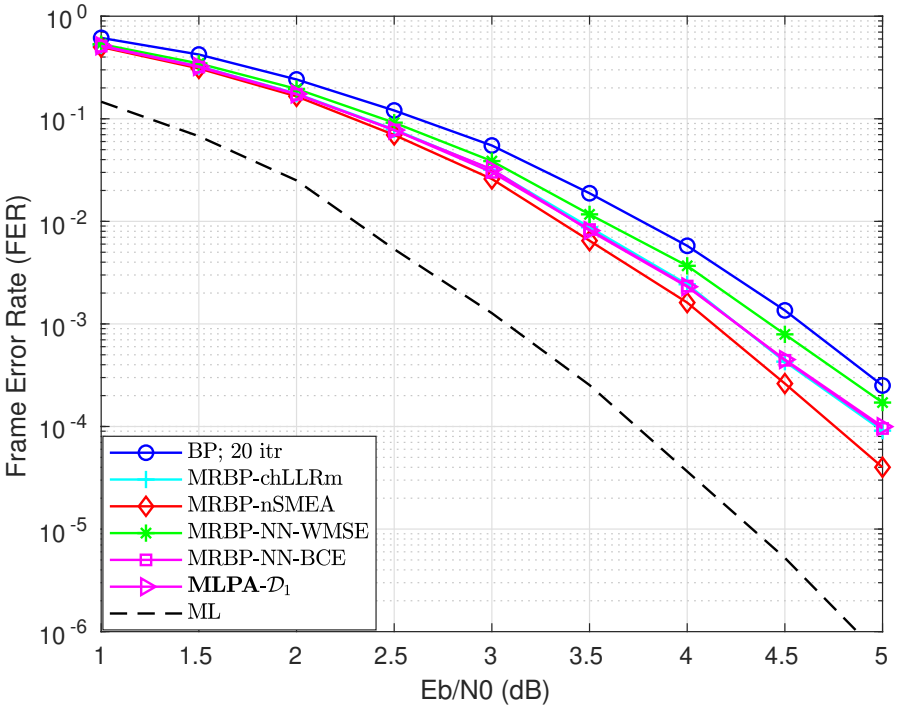


Figure 2.18: FER performance of an MRBP decoder with single-bit perturbations identified via a NN with different loss functions and compared to other expert-based rules.

To observe the effect of increasing the number of samples, in Fig.2.18, we also show the performance of the following model:

- **MLPA-$\mathcal{D}_1$**: Trained using the BCE loss but on a larger dataset consisting of 60M samples collected at 3 dB SNR following the transmission of randomly generated codewords. The model is optimized with the Adam optimizer, starting with a learning rate of 10^{-4}, which is reduced via a reduce-on-plateau scheduler based on validation performance. A dropout ratio of 0.1 is applied after the first two hidden layers. Training is performed with a batch size of 4096 for a total of 250 epochs.

As observed, **MLPA-$\mathcal{D}_1$** does not outperform its 1M-sample counterpart, suggesting that this MLP model, which features 46k trainable parameters on the TUKL(96,48) code, may have already reached its performance limit. Nevertheless, **MLPA-$\mathcal{D}_1$** serves as a baseline for the subsequent experiments. So, all the models in the coming experiments, are trained with a 60M-sample dataset.

How important is the model input?

Now that we have the baseline reference **MLPA-$\mathcal{D}_1$**, we proceed to evaluate the impact of the proposed input representation. As suggested at the beginning of the section, we consider as the model's input the pair $(\widetilde{|\mathbf{L}^{\text{ch}}|}, \mathbf{s})$ instead of $\tilde{\mathbf{y}}$. To this end, we train a new model:

- **MLPA-$\mathcal{D}_2$**: Trained on dataset $\mathcal{D}_2$ defined in (2.2), this model retains the same overall architecture as **MLPA-$\mathcal{D}_1$**, but its input dimension is increased from 96 to $2n - k = 144$. To ensure a fair comparison by keeping the total number of trainable parameters fixed at 46k, the size of the first hidden layer is reduced from $2n = 192$ to 155.

Fig. 2.19 compares the performance of **MLPA-$\mathcal{D}_1$** and **MLPA-$\mathcal{D}_2$** under the parallel MRBP decoding setup outlined earlier with $T = 5$ decoding attempts. The figure shows that training on $\mathcal{D}_2$ improves the FER performance compared to $\mathcal{D}_1$, confirming that providing the combination of $|\mathbf{L}^{\text{ch}}|$ and $\mathbf{s}^{\text{ch}}$ as input does contribute to increasing the model's ability to identify the VNs to perturb.

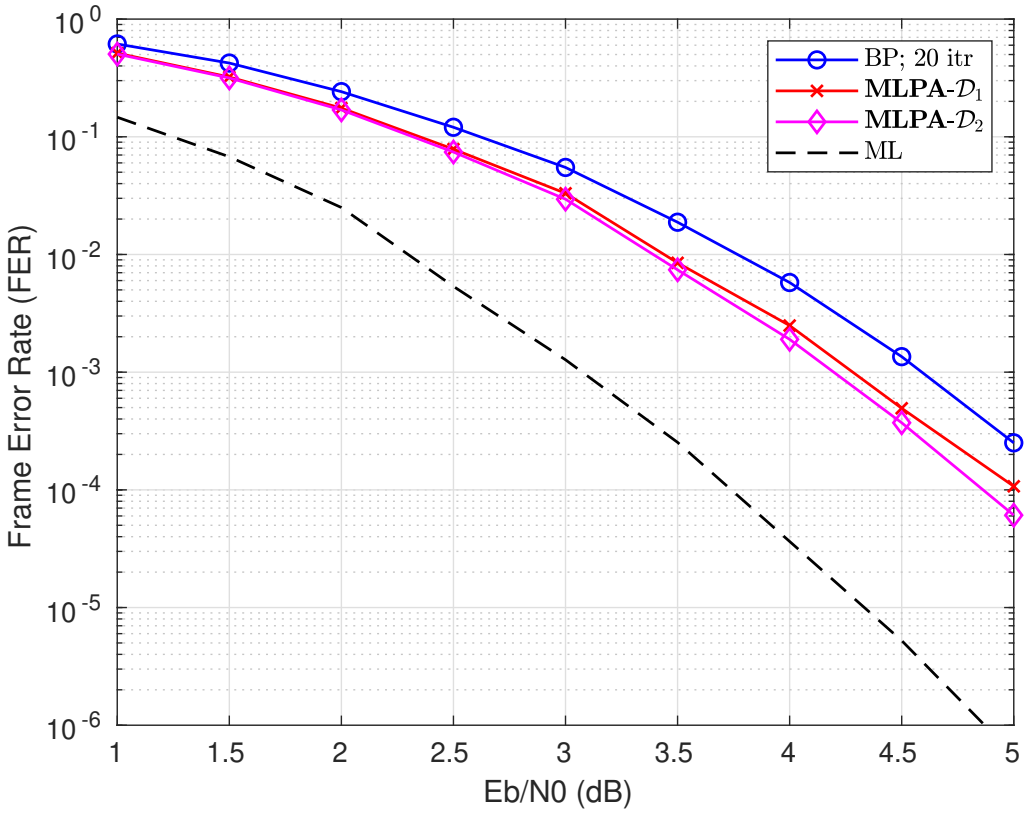


Figure 2.19: FER performance as function of E_b/N_0 for $T = 5$ MRBP decoding attempts, using different model inputs to select the VNs to perturb.

How important is the model architecture?

Having established that the pair $(\widetilde{|\mathbf{L}^{\text{ch}}|}, \mathbf{s}^{\mathbf{ch}})$ offers a more effective input representation, we next investigate whether a GRU-based architecture can surpass an MLP when trained on $\mathcal{D}_2$. For this purpose, we consider two models with approximately 20.6M parameters each:

- **GRU-$\mathcal{D}_2$**: A 5-layer stacked GRU with hidden size $6(2n - k)$, following the configuration in [1] and as implemented in section 1. The network runs for 5 time steps, with the final hidden state passed through a fully connected layer of size n and a sigmoid activation. Dropout of 0.1 is applied within the GRU layers.

- **MLPB-$\mathcal{D}_2$**: A custom MLP with seven hidden layers of 1835 neurons each, using ReLU activation and 0.1 dropout between layers, followed by an n-unit sigmoid output layer.

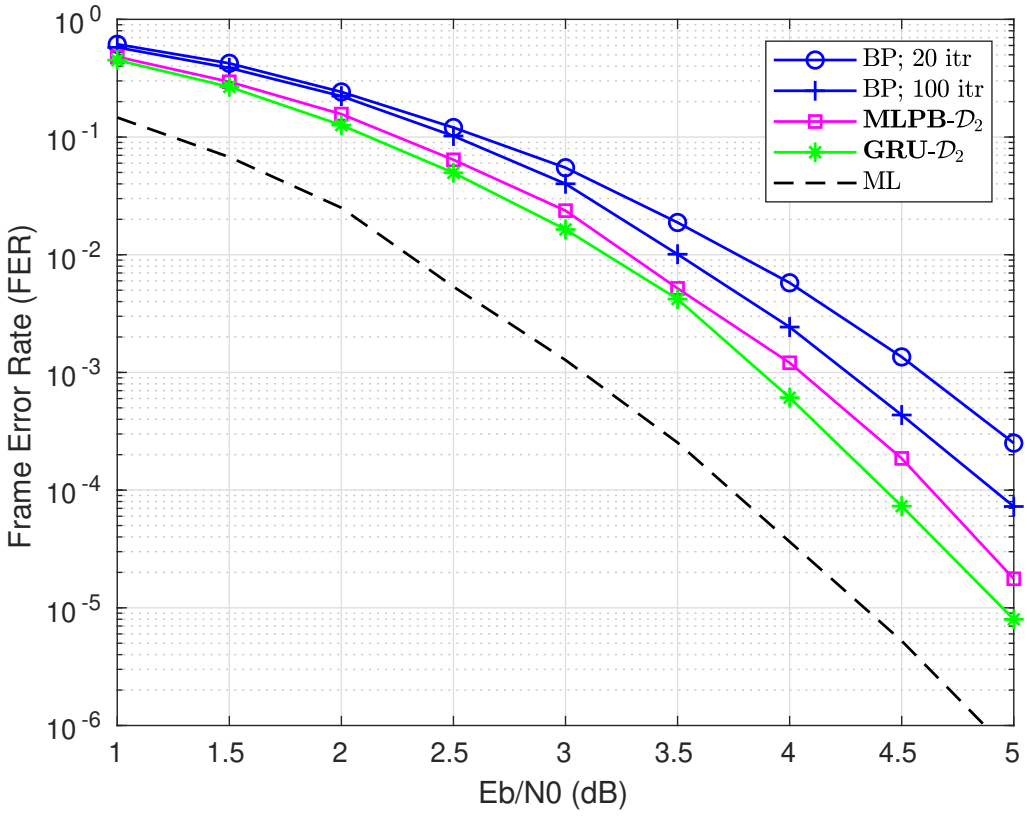


Figure 2.20: FER performance as function of E_b/N_0 for $T = 5$ MRBP decoding attempts, using different model architectures to select the VNs to perturb.

As shown in Fig. 2.20, the GRU model achieves better performance than the best-effort MLP in identifying problematic VNs for perturbation, consistent with the findings in [1]. Moreover, both models outperform BP decoding with 100 iterations, further demonstrating the ability of learned MRBP to enhance BP performance.

How does the new learned MRBP scheme compare to the expert rules?

This is perhaps the most important question and in fact the target of our studies in this chapter. We aim to investigate whether learning can indeed help more accurately select the VNs to perturb in MRBP. For that, we compare our best performing model **GRU-$\mathcal{D}_2$** to one of the best expert-based rule nSMEA. As depicted in Fig. 2.21, our trained model significantly outperforms nSMEA with just one perturbation, that is $T = 1$. This superior performance is maintained as more perturbations ($T = 5$, $T = 10$) are tested in parallel, indicating that learning-based methods can more accurately predict which bits to perturb in MRBP decoding. In Fig. 2.21, we also show the single-bit perturbation limit where a full search is performed by perturbing all VNs for each decoding failure. We observe that the learned model approaches faster the single-bit perturbation limit. On the other hand, running a 20.6M-parameter DNN for that task is significantly more complex than tracking oscillations in the messages exchanged during iterative decoding.

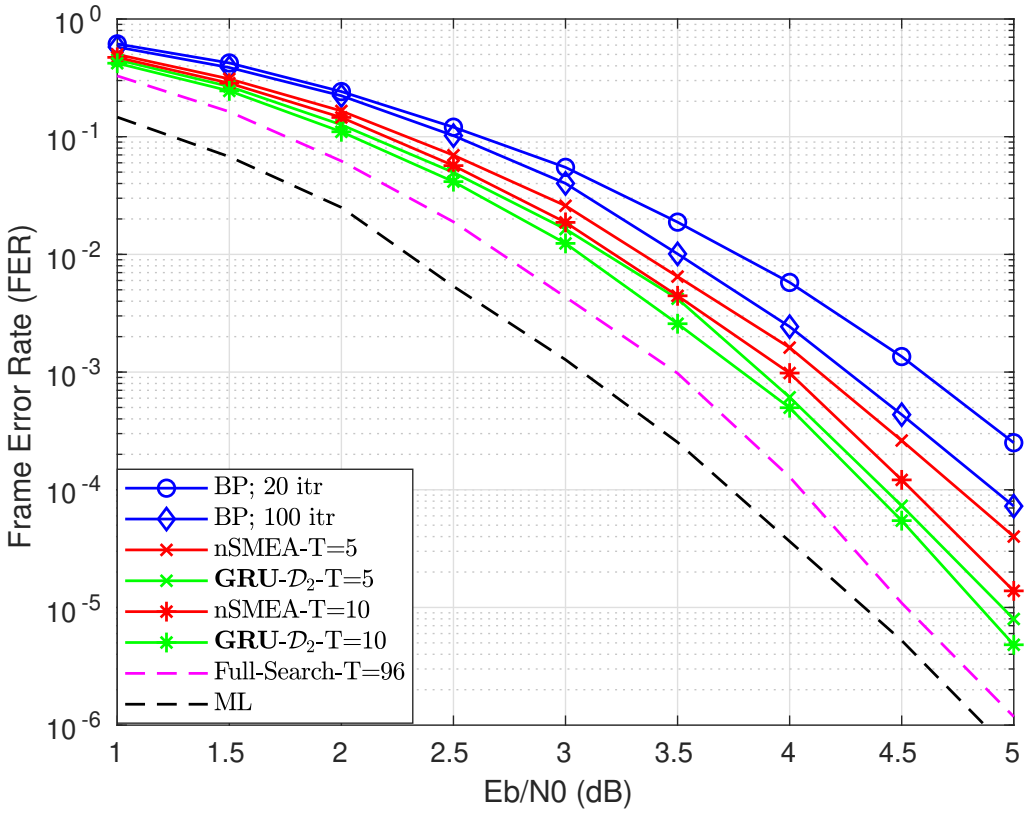


Figure 2.21: FER performance as function of E_b/N_0 for MRBP decoding using the model **GRU-$\mathcal{D}_2$** or the nSMEA expert rule to select the VNs to perturb.

How does the new learned MRBP scheme compare to other selected decoders?

We compare our **GRU-$\mathcal{D}_2$** learning-based MRBP decoder on the TUKL$(96, 48)$ code against two reference decoders: the After-burner from [38], and the EQML [39] which use different MRBP variations. As shown in Fig. 2.22, our model matches the performance of the After-burner with half the number of perturbations and surpasses EQML with the same number of perturbations. These results demonstrate that the proposed learning-based approach can both reduce the number of decoding attempts and enhance MRBP performance, compared to other existing solutions.

In Fig. 2.22, we also provide results for a GRU-based SBND model with 39M parameters, trained on the TUKL$(96, 48)$ code with 64M ML error patterns following the training methodology of Chapter 1. Compared to this standalone neural decoder, our **GRU-$\mathcal{D}_2$** learned MRBP decoder achieves superior performance with only half the number of parameters. In this case, it is more efficient to train an SBND model to supplement MRBP rather than using it as a standalone decoder.

Would the model benefit if we provide post-BP decoding information in input?

In both datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, the model input consists solely of channel-induced information and no explicit post-BP decoding information is provided. Except the implicit fact that the training samples correspond to failed BP attempts, the model has no direct knowledge of the decoder's internal state after BP terminates.

It is therefore natural to consider whether including post-BP information could help the model better identify the problematic VNs. To this end, we propose an input representation consisting of the nSMEA reliability measure of the VNs and the output syndrome vector $\mathbf{s}^{\text{app},(I_0)}$, which encodes the state of the check nodes (CNs) at the decoder's output in the final BP iteration I_0.

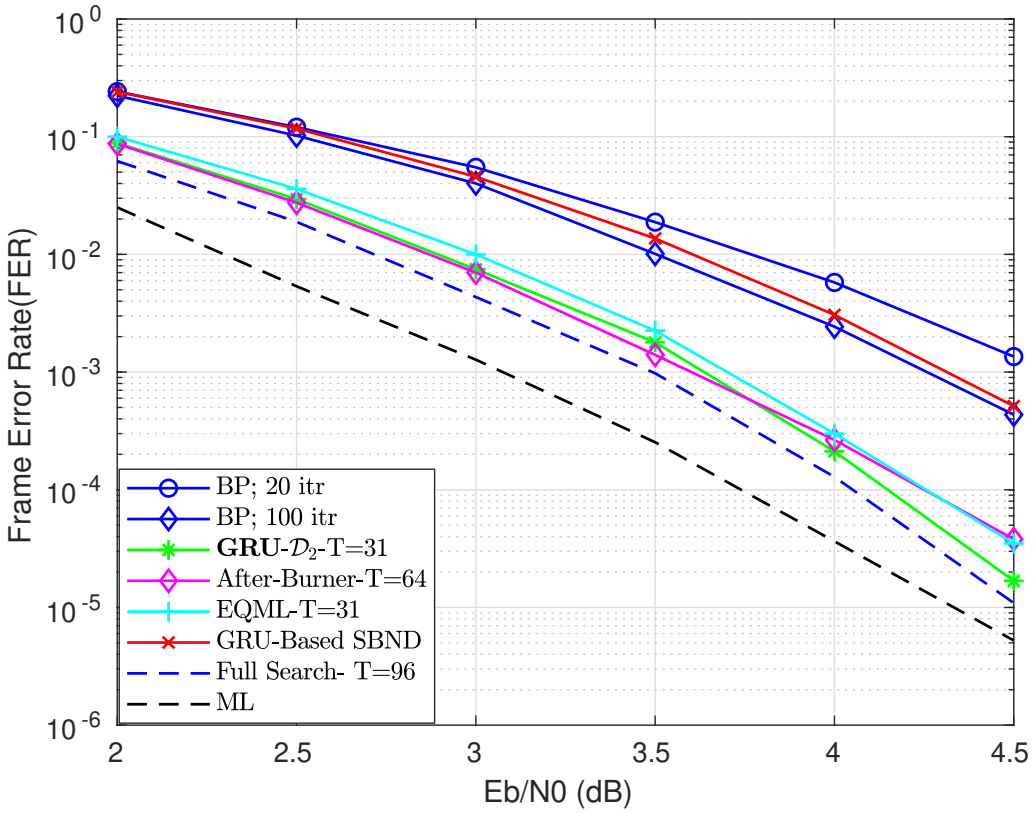


Figure 2.22: FER performance as function of E_b/N_0 for MRBP with **GRU-$\mathcal{D}_2$** model compared to other selected reference decoders on the TUKL(96, 48) code.

The **nSMEA** vector is normalized to the range $[0, 1]$ via

$$\widetilde{\mathbf{nSMEA}} = \frac{\mathbf{nSMEA}}{\max_i \mathbf{nSMEA}}. \tag{2.4}$$

The new dataset is then represented as:

$$\mathcal{D}_3 = \left\{ \left(\widetilde{\mathbf{nSMEA}}, \mathbf{s}^{\text{app},(I_0)} \right)_j, \mathbf{b}_j \right\}, \quad j \in \{1, \cdots, N_{tr}\}. \tag{2.5}$$

To evaluate the performance of this input representation, we trained the following model:

- **GRU-$\mathcal{D}_3$**: This model features the same GRU-based model **GRU-$\mathcal{D}_2$**, trained under the same setup but on $\mathcal{D}_3$

We compare the performance of this model to the **GRU-$\mathcal{D}_2$** in Fig. 2.23 for $T = 5$ decoding attempts. While the model trained on $\mathcal{D}_3$ achieves better performance than the nSMEA criterion, its performance in the low-SNR region is comparable to that of the model trained on $\mathcal{D}_2$. However, at higher SNRs, the **GRU-$\mathcal{D}_3$** model starts to lag behind.

A natural test is then to train the same architecture on $\mathcal{D}_3$ using data collected from the higher-SNR region. Specifically, we have trained the **GRU-$\mathcal{D}_3$** model on BP failures collected at an SNR of 4dB. Surprisingly, this did not yield any improvement (results not shown here), suggesting that the limitation may arise from how meaningful the information is when combining nSMEA with the output syndrome, particularly in the high-SNR regime.

While we do not have a definitive explanation for this trend, we can hypothesize the following. At high SNRs, the dominant error patterns are unstable oscillating errors. Although nSMEA is generally a good proxy for identifying oscillation-based errors, its ability to distinguish between different oscillating

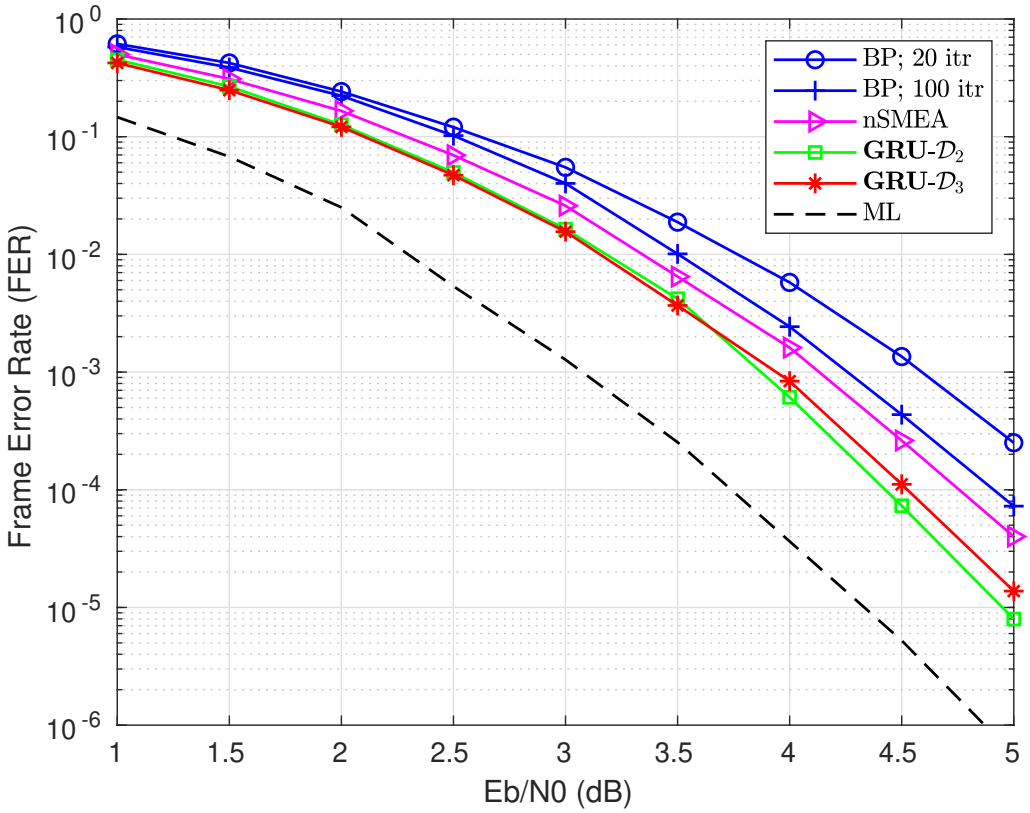


Figure 2.23: FER performance as function of E_b/N_0 for MRBP with post vs pre-BP model input representations with $T = 5$.

patterns becomes limited in this regime. The output syndrome, on the other hand, is an instantaneous snapshot that strongly depends on the iteration count and does not capture the full trajectory of the decoding process. Both metrics ultimately rely on the state of the BP decoder, which has failed, thereby limiting the reliability of these features.

It is noteworthy here, that we have tried to combine other post-BP metrics to give in input to the model but none achieved performance better than $\mathcal{D}_3$.

Does this approach scale?

By leveraging SBND, we have shown that learning can indeed help identify the VNs to perturb in MRBP more accurately than the expert rules, reducing the number of decoding attempts required while improving performance. On the other hand, **the current model size is already very large and impractical for deployment**, raising questions about scalability.

For instance, we trained a GRU-based model with approximately 36M trainable parameters on the CCSDS(128,64) code using 80M samples collected according to $\mathcal{D}_2$. The FER results are reported in Fig. 2.24. **Despite its large size, the model no longer outperform MRBP using the nSMEA criterion on this particular example**, highlighting the need for even more sophisticated architectures and larger training datasets as code length increases.

2.5 Conclusion

In this chapter, we investigated multi-round BP (MRBP) decoding as a powerful strategy to push BP closer to MLD performance but highlighted that MRBP requires a large number of decoding rounds to achieve this goal. This limitation stems from relatively weak variable-node (VN) selection criteria,

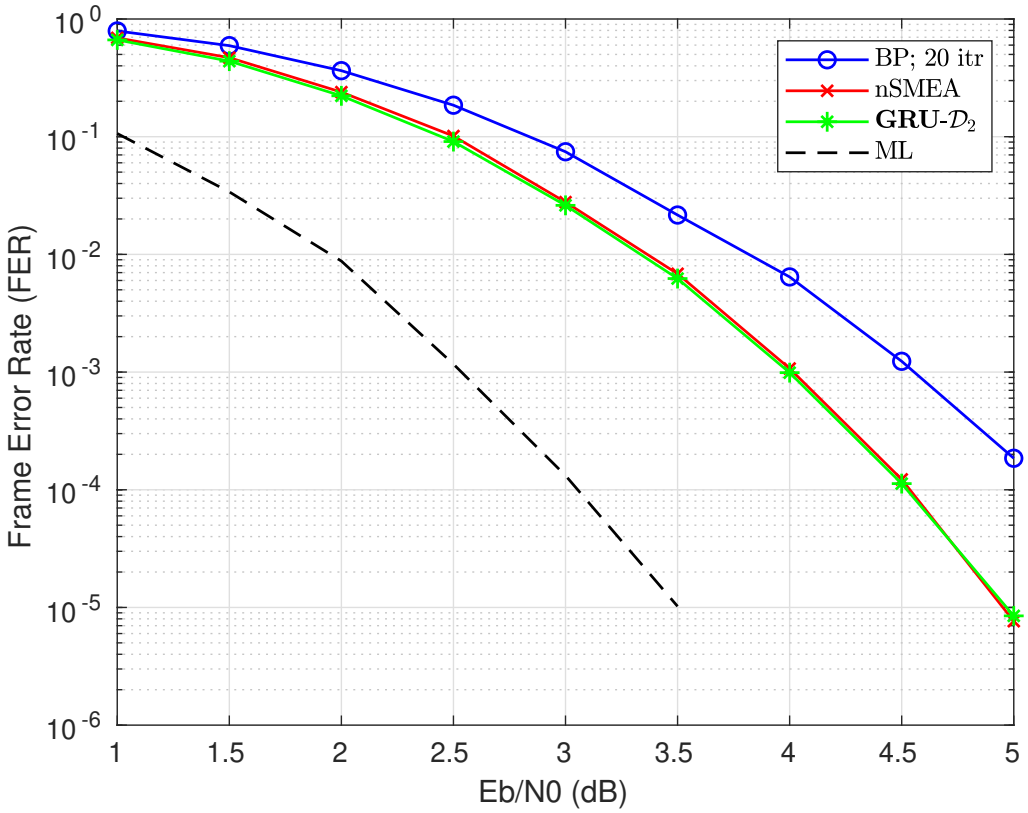


Figure 2.24: FER performance as a function of E_b/N_0 for MRBP with **GRU**-$\mathcal{D}_2$ trained on 80M samples from the CCSDS(128,64) code with $T = 5$ decoding attempts.

which reduce the effectiveness of identifying and perturbing the most problematic VNs that hinder BP's convergence.

Motivated by this observation, we first explored a first learning-based approach, proposing lightweight machine learning models to fuse multiple expert-designed VN selection rules into a more effective strategy. Although the results were underwhelming, this investigation helped identify potential weaknesses in our methodology and guided the deeper exploration carried out in the following chapter.

Then, we extended our exploration of how learning can augment MRBP decoding, this time employing more powerful tools by establishing a connection to SBND. Specifically, the problematic bits targeted for perturbation in MRBP correspond to a subset of channel errors that SBND is particularly effective at estimating. Leveraging this insight, we trained learned models to identify which bits to perturb, demonstrating that these models can outperform the expert-designed rules in selecting the most critical bits. Consequently, the number of decoding rounds required by MRBP to approach MLD performance was reduced. Despite this success, a key limitation remains: the current learned models are too large for practical deployment, highlighting the need for more lightweight, dedicated architectures.

In our work on NN-aided MRBP, we adopted an input representation consisting of the pair $(\mathbf{L}^{\text{ch}}, \mathbf{s})$. This representation serves as a sufficient statistic for maximum-likelihood decoding and has proven effective in SBND, particularly when the targets were defined as ML error patterns. In the MRBP setting, however, the goal is to identify bits to perturb that would enable BP decoding to succeed. While this *objective* representation performed reasonably well, it provides no explicit information about the current decoder state. We experimented with incorporating *subjective* post-BP metrics through the pair $(\mathbf{nSMEA}, \mathbf{s}^{\text{app},(I_0)})$, which, although outperforming expert rules, still fell short of the performance achieved by the pre-BP input representation. Moreover, this representation captures only a coarse approximation of the decoder's internal state and lacks fine-grained insight into BP's

iterative behavior. These observations highlight a clear opportunity for further investigation into richer and more informative input representations that could better guide the model in identifying the bits most critical for improving BP convergence.

We believe that the model would benefit most if it were provided with a more comprehensive view of the decoding process, allowing it to learn from the full behavior of the BP decoder. While directly capturing the entire decoding trajectory is costly, especially for larger codes, another idea is to run the model in parallel with the BP decoder. At each decoding iteration, the model could be fed with relevant features summarizing the current decoder state, enabling it to refine its predictions iteratively until reaching a confident decision about which bits are most likely responsible for BP's failure. Further investigation is needed to design suitable model architectures capable of processing this sequential information effectively, with recurrent neural networks, such as the stacked GRU, being a strong candidate due to their ability to model temporal dependencies. Yet our first investigations in this direction have not been successful so far. We believe that the model would benefit most if it were provided with a more comprehensive view of the decoding process, allowing it to learn from the full behavior of the BP decoder. While directly capturing the entire decoding trajectory is costly, especially for larger codes, another idea is to run the model in parallel with the BP decoder. At each decoding iteration, the model could be fed with relevant features summarizing the current decoder state, enabling it to refine its predictions iteratively until reaching a confident decision about which bits are most likely responsible for BP's failure. Further investigation is needed to design suitable model architectures capable of processing this sequential information effectively, with recurrent neural networks, such as the stacked GRU, being a strong candidate due to their ability to model temporal dependencies. Yet our first investigations in this direction have not been successful so far.

The NN-aided MRBP approach presented in this work was limited to single-bit perturbations, primarily for simplicity. Extending this approach to multiple-bit perturbations introduces significant combinatorial complexity. For instance, if we were to consider perturbing up to two bits simultaneously using the same training data generation strategy, then for the TUKL$(96, 48)$ code, the number of possible bit combinations would be $\binom{96}{2} = 4560$. This means that the model would need to evaluate whether each of these 4560 combinations leads to successful decoding, significantly increasing the difficulty of the learning task. An alternative is to adopt more sophisticated learning paradigms that avoid exhaustive enumeration. For instance, reinforcement learning could train a model to make sequential bit-perturbation decisions based on decoder feedback, effectively learning a policy to navigate the combinatorial space. This approach can align naturally with dynamic perturbation strategies originally used in the augmented BP decoder.

3 Improving Neural BP Decoders via Diversity and Post-Processing

3.1 Motivation

In this section, we focus on improving the decoding performance of short LDPC codes, by leveraging on and combining several techniques, such as neural belief-propagation (BP), decoding diversity, and ordered statistics decoding (OSD) post-processing.

It is worth noticing that first works on improving the BP decoding performance actually focused on so-called high-density parity-check (HDPC) codes [40–43]. These codes are defined by higher density bipartite graphs – *e.g.*, Bose-Chaudhuri-Hocquenghem (BCH) codes, Reed-Muller codes, binary Golay codes, etc. – for which the standard iterative BP decoding usually yields very poor error correction performance. In [40], an adaptive BP decoding approach was proposed, in which the decoding graph used by BP is updated at each iteration, according to the output of a reliability-based decoding algorithm, such as (OSD) [44]. The random recurrent decoding in [41, 42] and the multiple-bases BP decoding in [43] exploit a decoding diversity approach, in the form of different graph representations of the code, implying several BP decoders working either in serial or in parallel.

More recently, deep neural networks have received significant interest for improving the decoding performance of short codes [45–50]. A weighted BP decoding has been introduced in [45], where the weights are optimized using a neural network (NN). The topology of the NN mimics the BP decoding process, with unwrapped decoding iterations. The approach can be used with either a feedforward (FF) or a recurrent NN (RNN), the corresponding decoders being termed as BP-FF or BP-RNN. It has been shown in [46] that the BP-RNN is able to outperform the standard BP decoder for short BCH codes, belonging to the class of HDPC codes. Subsequently, several variants of NN-based BP decoding have been proposed in the literature. In [49], a pruning method of irrelevant check-nodes in a neural BP model has been proposed, aimed at jointly optimizing the code construction and the decoding. The design of new decoding rules for finite-alphabet iterative decoders (FAIDs), based on a quantized NN model, was proposed in [47]. Moreover, a new training method of the quantized NN model was introduced in [48], where training sets are constructed by sampling errors with trapping set support, to achieve decoding diversity for FAIDs on the binary symmetric channel. Resolving decoding failures due to trapping sets, by means of deep learning techniques, has also been recently investigated in [51].

Here, we consider the BP-RNN decoding of short LDPC codes. One critical difficulty faced by BP decoding in the finite blocklength regime is the presence of particular structures in the bipartite graph, which prevent the decoding algorithm from converging. Examples of such structures are trapping sets [52] and absorbing sets [53], and they are closely related to the pseudo-codewords [54] and near-codewords [55] concepts. We shall focus on the absorbing set concept, introduced in [53] as a combinatorial object associated with the bipartite graph, and defined independently of the particular message-passing decoding or channel noise model. For long LDPC codes, such structures are known to be responsible of the so-called error floor phenomenon, since their size may be relatively small with respect to the length of the code [53]. However, for short codes, their size may be comparable to the number of errors that the code must correct, therefore possibly inducing a significant degradation of the error correction performance in the waterfall region. To address this issue, we take a decoding diversity approach, implying several BP-RNN decoders working either in serial or in parallel, where each BP-RNN is trained to decode errors corresponding to absorbing sets of a specific type. We further combine our approach with a low-order OSD post-processing step, providing an efficient way to bridge the gap to maximum likelihood (ML) decoding. The main contributions of our work are summarized below.

Absorbing set classification and specialization of BP-RNN decoding. We first propose a graph-search based algorithm, combining backtracking and a deep-first search like procedure, to enumerate, in an efficient way, all the absorbing sets of a given size in the bipartite graph. We then perform a fine classification of the enumerated absorbing sets, according to the degree profile of the check-nodes in the induced sub-graph, and train a specific BP-RNN decoder for each absorbing set class.

BP-RNNs selection and decoding architectures. To reduce the number of the BP-RNN decoders, we propose a selection procedure, where the selected decoders are the most complementary in terms of the errors they can decode. We then consider two decoding architectures, in which the selected BP-RNN decoders are executed in an either parallel or serial manner, and define the appropriate metrics to assess their computational complexity and decoding latency.

BP-RNN diversity with OSD post-processing. We further combine our approach with an OSD post-processing step, applied in case none of the selected BP-RNN decoders outputs a codeword. There are two motivations behind the use of such a post-processing step. First, for short LDPC codes, the OSD complexity is limited, compatible with practical applications, especially when the OSD order is small. Here, we restrict the order of the OSD post-processing step to either 0 or 1. Second, it may benefit from the diversity brought by the use of multiple BP-RNN decoders. Indeed, we show that the coding gain brought by the use of multiple BP-RNN decoders is actually amplified by the use of the OSD post-processing, resulting in a significant improvement of the error correction performance.

Regular BP with OSD post-processing. Finally, we also consider the case when OSD post-processing is applied after the standard BP decoder, with the aim of proposing an alternative solution (and therefore a different performance/complexity trade-off) to the previous BP-RNN diversity based approach. The soft input of the OSD post-processor is defined as a weighted sum of a posteriori LLRs across the BP decoding iterations, which can be conveniently modeled and optimized by a simple neural approach. In the process, we also propose the use of the focal loss function [56], which is shown to be better suited for this specific task. To approach the maximum-likelihood decoding performance, we further consider a multiple OSD post-processing, where each OSD processes either the above weighted (neural) sum, or the a posteriori LLRs of the BP at some specific (carefully chosen) decoding iterations. We also propose a method to reduce the number of candidate codewords in the OSD post-processing step, and show that the proposed approach is scalable for long LDPC codes.

Related publications. The above contributions have been published in:

[RMSF21] J. Rosseel, V. Mannoni, V. Savin, and I. Fijalkow, "Error structure aware parallel BP-RNN decoders for short LDPC codes" in Proc. of *International Symposium on Topics in Coding (ISTC)*, August 2021.

[RMFS22] J. Rosseel, V. Mannoni, I. Fijalkow, and V. Savin, "Decoding short LDPC codes via BP-RNN diversity and reliability-based post-processing", *IEEE Transactions on Communications*, vol. 70, no. 12, pp. 7830–7842, 2022.

[RMSF23] J. Rosseel, V. Mannoni, V. Savin, and I. Fijalkow, "Sets of complementary LLRs to improve OSD post-processing of BP decoding" in Proc. of *International Symposium on Topics in Coding (ISTC)*, September 2023.

We provide an extended summary of the above contributions in the next sections. For more details we refer to the above publications.

3.2 BP-RNN Diversity from Absorbing Sets Classification: Training, Selection, and Decoding Architectures

We consider a *decoding diversity* approach, implying several BP-RNN decoders working either in serial or in parallel, where each BP-RNN is trained to decode errors corresponding to absorbing sets of a specific type. To do so, we need:

- An efficient algorithm capable to enumerate the absorbing sets of the Tanner graph (up to a given size),
- A classification method (grouping in the same class absorbing sets with similar structure),
- A method to construct a specific training set for each class of absorbing sets (thus providing a BP-RNN decoder specialized on the class),
- A method to select a small number of specialized BP-RNN decoders (to reduce the overall decoidng complexity),
- A method to organize the BP-RNN diversity, *i.e.*, in an either parallel or serial architecture.

We describe the above steps in the subsequent sections, then provide numerical results and discuss the implications of the proposed approach.

3.2.1 Absorbing sets: search algorithm and classification

A brute-force search algorithm to enumerate all the absorbing sets of a given size ν would have to explore all the $\binom{N}{\nu}$ candidates, where N is the code length, which may become computationally intractable even for relatively small values of N and ν. Several exhaustive/non-exhaustive enumeration algorithms have been proposed in the literature, to enumerate *elementary* trapping sets [57–59], trapping or absorbing sets for specific classes of LDPC codes [60, 61] or only small or dominant such structures [62, 63], and *fully* absorbing sets [64, 65]. Several of these algorithms rely on a linear programming based branch-and-bound approach, *e.g.*, [59, 64, 65].

In [RMFS22], we propose a graph search based algorithm, which, to the best of our knowledge, is the first specifically developed for the exhaustive enumeration of absorbing sets, without any restriction on the structure of the Tanner graph of the absorbing sets to be enumerated. The proposed algorithm is essentially a backtracking algorithm that incrementally builds absorbing set candidates, and abandons a candidate as soon as it determines that it cannot possibly be completed to an absorbing set. Candidates are built incrementally by traversing the bipartite graph in a depth-first search (DFS) manner, that is, starting from a variable-node chosen as root, and exploring as deeply as possible before backtracking. The main difference with the standard depth-first search is that our algorithm does not visit only one, but a subset of variable-nodes at each depth level, which are chosen to increment the candidate solution (for details, see [RMFS22]).

To illustrate the capability of the proposed algorithm, we consider the following two codes, both of rate 1/2, which will be subsequently used throughout this section.

Code-1 is a regular LDPC code, of length 64 bits, with variable-nodes of degree 3 and check-nodes of degree 6, constructed by using the progressive edge growth (PEG) algorithm [66]. It has girth $g = 6$, with multiplicity 164 (*i.e.*, number of cycles of length g).

Code-2 is the LDPC code from [67], known as the CCSDS LDPC code. It is of length 128 bits, with half of variable-nodes of degree 3 and the other half of degree 5, and check-nodes of degree 8. It has girth $g = 6$, with multiplicity 2336.

Table 3.1: Number of Extended-Types (ET) and Absorbing Sets (AS)

	Code-1		Code-2	
ν	ET-Number	AS-Number	ET-Number	AS-Number
3	1	164	1	32
4	2	1 452	6	944
5	3	9 413	12	11 504
6	9	64 813*	32	152 824
7	16	450 340	69	2 124 928
8	24	2 994 834*	157	28 670 736

*Code-1 contains one codeword of weight 6, and 37 codewords of weight 8, corresponding to absorbing sets with $\omega = 0$

In Table 3.1, we provide the total number of absorbing sets ("AS-Number" columns), for both Code-1 and Code-2, and absorbing set size values $\nu \leq 8$. We note that for Code-2 and $\nu = 8$, a brute-force search algorithm would require exploring a number of $\binom{128}{8} \approx 2^{40}$ candidates. Our algorithm enumerated all the absorbing sets of size $\nu = 8$ in 38 minutes (Intel Xeon @2.20GHz processor).

The type of an absorbing set A is defined as ν-(ω, ε), where $\nu = |A|$ is the number of variable-nodes in A, and $\omega := |O(A)|$, resp. $\varepsilon := |E(A)|$, is the number of checks of degree odd, resp. even, in the subgraph induced by A. Thus, the subgraph induced by A comprises ν variable-nodes and $\omega + \varepsilon$ check-nodes. However, note that absorbing sets of the same type do not necessarily induce the same subgraph. To further characterize the structure of the induced subgraph, we define the check-node degree profile (of the subgraph induced by A) as $P_c = (m_1, m_2, \dots)$, where m_d is the number of check-nodes connected to exactly d variable-nodes in A (note that the sequence m_d is actually finite, of length equal to the maximum check-node degree in the subgraph induced by A). Finally, we define the *extended type* of A as ν-$(\omega, \varepsilon, P_c)$.

We classify the absorbing sets according to their *extended type*. The number of different extended-type values (*i.e.*, number of different classes) is given in Table 3.1, in the "ET-Number" column.

3.2.2 Specialization of BP-RNN decoding

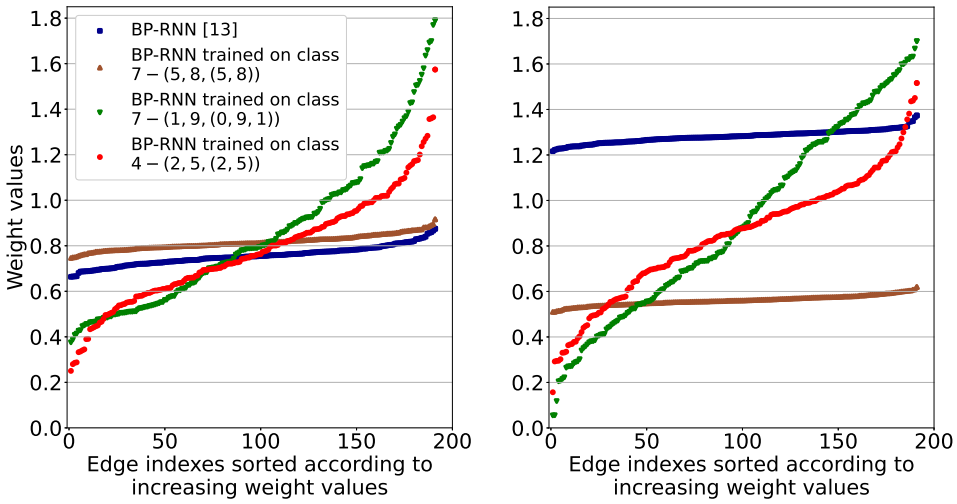
Having at our disposal an efficient algorithm to enumerate absorbing sets of a given size, we perform a fine classification of the found absorbing sets, by grouping into a same class absorbing sets with the same extended type. The goal is to train a specific BR-RNN decoder for each class. The approach bears similarity to, and is motivated by [48, 68], where decoding rules for FAIDs have been either designed or learned to correct specific trapping sets for the binary symmetric channel.

We consider a binary-input AWGN channel, with BPSK alphabet (± 1) inputs, and noise variance σ^2. Since both the channel and the BP-RNN decoder are symmetric, we can train the latter on noisy versions of the all-zero codeword.

One way to generate a training set for a given class of absorbing sets, *i.e.*, a given extended type ν-$(\omega, \varepsilon, P_c)$, is to randomly generate noisy versions of the all-zero codeword, and to select those whose error support is an absorbing set having the desired extended type. This would be rather tedious and time consuming. A more efficient way is to use our knowledge of absorbing sets (determined by the algorithm from previous section), and generate Gaussian noise by means of a truncated Gaussian distribution, to produce errors only on desired locations. Precisely, we proceed as follows. We first chose an absorbing set A at random, from those having the desired extended type ν-$(\omega, \varepsilon, P_c)$. Then we generate a random word $y = (y_n = 1 + z_n)_{n=1,\dots,N}$, with

$$z_n \sim \begin{cases} \mathcal{N}(0, \sigma^2, -\infty, -1), & \text{if } n \in A \\ \mathcal{N}(0, \sigma^2, -1, \infty), & \text{otherwise} \end{cases} \tag{3.1}$$

where $\mathcal{N}(0, \sigma^2, a, b)$ denotes the truncated normal distribution with mean 0 and variance σ^2, taking values in the interval (a, b). The training set is obtained by repeating the above procedure (including



(a) Data pass layer

(b) A posteriori layer

Figure 3.1: Weight profiles of various BP-RNN decoders (Code-1, SNR = 5 dB)

the random choice of A) multiple times. This way, the training set is representative of the *error class* (identified by the extended type of the absorbing error set), and thus the trained BP-RNN decoder becomes specialized in decoding the errors of the class.

To illustrate the specialization of the trained decoder, Fig. 3.1 shows the weight values for three BP-RNN decoders, trained for three different error classes, with corresponding extended types shown in the legend. The (unspecialized) BP-RNN decoder trained as in [46] is also shown. To avoid clutter, weight values are sorted in increasing order, and an ordered set of weight values is referred to as *weight profile*[1]. We consider the Code-1 and show the weight profiles for the both data pass and a posteriori layers. (See [RMFS22] for the definition of data pass and a posteiori layers.) The weight profiles in Fig. 3.1 clearly indicate that weight optimization responds differently to training sets corresponding to different error classes. We exploit in the next section the diversity induced by the BP-RNN specialization.

3.2.3 BP-RNN diversity selection

Using the procedure from previous section, we train one specialized BP-RNN decoder for each error class, *i.e.*, extended type[2] ν-$(\omega, \varepsilon, P_c)$, with absorbing error set size $\nu \leq \nu_{\max}$ (the choice of $\nu_{\max}$ is discussed in Section 3.2.5, providing the numerical results). Let J denote the total number of specialized BP-RNN decoders.

Here, we propose a selection procedure to reduce the number of BP-RNN decoders, as well as to avoid similar training effects (which may arise, for instance, due to absorbing sets of a given size containing absorbing sets of smaller sizes). To do so, we assess the complementarity of the trained decoders, in terms of the errors they can decode.

We first generate a common test set, T, containing random noisy versions of the all-zero codeword, which is then used to assess the individual error correction performance of each of the trained decoders. We denote by $F_j \subset T$ the susbset of words on which the BP-RNN decoder D_j failed, where $j = 1, \ldots, J$. Then, we recursively construct an *ordered list* of decoder indexes, denoted $\mathfrak{J}$, as follows. We start by

[1]Note that different weight profiles indicate different sets of weight values, while similar weight profiles indicate similar sets of weight values. However, in the latter case, similar weights may apply to different edges.

[2]We take off extended types for which $\omega = 0$, as they correspond to the support of non-zero codewords. Such errors cannot be detected or corrected.

initializing $\mathfrak{J}$ as the empty list, $\mathfrak{J} = \emptyset$. To add a new index, j_{new} to $\mathfrak{J}$, we use the following rule,

$$j_{\text{new}} = \underset{j \in \{1,\dots,J\} \setminus \mathfrak{J}}{\operatorname{argmin}} |F_{\mathfrak{J}} \cap F_j| \,, \tag{3.2}$$

where $F_{\mathfrak{J}} := T$, if $\mathfrak{J} = \emptyset$, and $F_{\mathfrak{J}} := \cap_{i \in \mathfrak{J}} F_i$, otherwise. The above rule is applied recursively J times, until $\mathfrak{J}$ contains all the decoder indexes $j = 1, \dots, J$, in a sorted order. In case the argmin in (3.2) is not unique, an arbitrary one is chosen.

Note that when $\mathfrak{J} = \emptyset$, the rule (3.2) rewrites as $j_{\text{new}} = \operatorname{argmin}_{j \in \{1,\dots,J\}} |F_j|$. Hence, the first decoder (index) added to the list is the one minimizing the word error rate. Subsequently, when $\mathfrak{J} \neq \emptyset$, the new decoder added to the list is the most *complementary* with those already in $\mathfrak{J}$, in the sens that it minimizes the number of words on which all the decoders indexed by $\mathfrak{J} \cup \{j_{\text{new}}\}$ fail.

For $Z \leq J$, let $\mathfrak{J}(1{:}Z) \subset \mathfrak{J}$ be the sublist defined by the first Z indexes in $\mathfrak{J}$. We define

$$\mathcal{D}_Z := \{D_j \mid j \in \mathfrak{J}(1{:}Z)\} \tag{3.3}$$

Note that $\mathcal{D}_Z$ is an *ordered list of decoders*[3], which we will refer to as *BP-RNN diversity* of size Z (using a similar terminology to the one in [48,68]). The Z BP-RNN decoders in $\mathcal{D}_Z$ may then be used with either a parallel or a serial decoding architecture, as discussed in the next section. The value of Z may be dictated by complexity reasons, or chosen to ensure small (negligible) degradation of the error correction performance, with respect to the case when all J BP-RNN decoders are used.

3.2.4 BP-RNN diversity decoding architectures

We consider a BP-RNN diversity $\mathcal{D}_Z$, comprising Z BP-RNN decoders. For simplicity, we index the BP-RNN decoders in $\mathcal{D}_Z$ from 1 to Z, thus $\mathcal{D}_Z = \{D_1, \dots, D_Z\}$. Fig. 3.2 shows the proposed parallel and serial architectures, using the Z BP-RNN decoders in $\mathcal{D}_Z$.

In the parallel architecture, each decoder outputs an estimate $\hat{c}_j = (c_{j,1}, \dots, c_{j,N}) \in \{0,1\}^N$ of the transmitted codeword c, according to the sign of the corresponding LLR values at the last decoding iteration. The output of the parallel architecture is determined as the maximum likelihood (ML) codeword, among the codewords outputted by the constituent BP-RNN decoders (if any, see below). For the binary-input AWGN channel (with ± 1 inputs), the ML criterion simply writes as

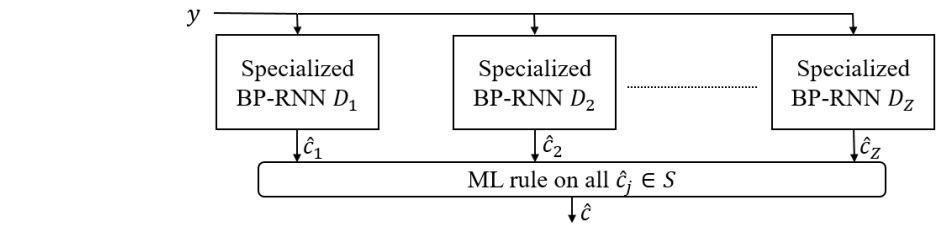
$$\hat{c} = \underset{\hat{c}_j \in S}{\operatorname{argmax}} P(\hat{c}_j | y) = \underset{\hat{c}_j \in S}{\operatorname{argmin}} \sum_{n=1}^{N} y_n \hat{c}_{j,n}, \tag{3.4}$$

where $S := \{\hat{c}_j \mid \text{syndrome}(\hat{c}_j) = 0\}$ denotes the set of $\hat{c}_j$'s verifying the syndrome. Decoding is then successful if $\hat{c}$ is equal to the transmitted codeword. If none of the BP-RNN decoders outputs a codeword, the decoding fails. In such a case, the ML criterion – or a similar bitwise maximum a posteriori criterion – may still be applied to select one of the $\hat{c}_j$ outputs, if desired, *e.g.*, in order to minimize the bit error rate of the decoder (however, we will only be concerned with word error rate results in this paper).

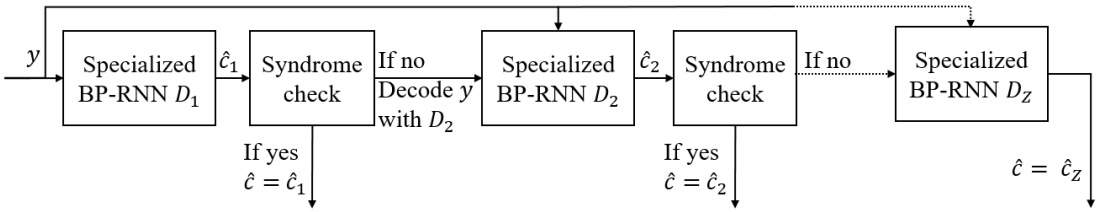
In the serial architecture, the constituent BP-RNN decoders are run sequentially according to the order given by the sorting procedure from Section 3.2.3. Decoding stops as soon as a BP-RNN decoder D_j outputs a codeword $\hat{c}_j$ (syndrome$(\hat{c}_j) = 0$), which becomes the output $\hat{c}$ of the serial architecture. Decoding is then successful if $\hat{c}$ is equal to the transmitted codeword. If none of the BP-RNN decoders outputs a codeword, decoding fails. For simplicity, in Fig. 3.2b, we take $\hat{c}_Z$ as the output of the serial architecture in this case.

The parallel architecture yields a reduced maximum decoding latency, compared to the serial one. This comes however at the cost of an increased complexity, from the both computational and hardware perspectives, since Z BP-RNN decoders must be instantiated in hardware. For the serial architecture,

[3] As a matter of fact, the three BP-RNN decoders in Fig. 3.1, specialized on absorbing set error classes, correspond to $\mathcal{D}_{Z=3}$.



(a) Parallel architecture



(b) Serial architecture

Figure 3.2: BP-RNN diversity decoding architectures

only one decoder may be instantiated in hardware, which may then be reused to perform sequentially the Z BP-RNNs (while updating the corresponding set of weights). For details on the average-case computational complexity, as well as the average-case decoding latency, for the both parallel and serial architectures, we refer to [RMFS22].

3.2.5 Numerical results

Training settings

We consider the two LDPC codes of rate 1/2 and length either 64 (Code-1) or 128 (Code-2) bits, detailed in Section 3.2.1. We train one specialized BP-RNN decoder for each absorbing set error class, with error set size $\nu \leq \nu_{\max}$.

- For Code-1, we choose $\nu_{\max} = 8$, which gives a total number of $J = 52$ error classes (see Table 3.1 [4]).
- For Code-2, we choose $\nu_{\max} = 7$, which gives a total number of $J = 120$ error classes (see Table 3.1).

For Code-2, the choice of $\nu_{\max} = 7$ is due to complexity reasons (to limit the number of trained decoders). However, we note that for an SNR $= 4$ dB, in the waterfall region of Code-2, the average number of errors is $Np_e = 7.2$, where $N = 128$ is the code-length, $p_e = Q(1/\sigma) = 0.0565$ is the error probability of the binary-input AWGN channel, σ is the standard deviation of the Gaussian noise, and Q denotes the Q-function. For Code-1, for the same SNR $= 4$ dB, the size of a random error set is less than or equal to the chosen $\nu_{\max} = 8$, with probability slightly greater than 0.99.

Each specialized BP-RNN is trained independently, using the training set construction method presented in Section 3.2.2. In addition, we also train an unspecialized BP-RNN decoder, according to the procedure described in [46], to provide a benchmark for the presented numerical results. We use the same SNR for training and testing, thus, all BP-RNNs are trained for each SNR value ranging from 1 dB to 6 dB, with a step of either 0.5 dB or 1 dB. Finally, we mention that we used the Keras library for training, with training parameters shown in Table 3.2.

[4] Note that the total number of error classes (ET-Number) in Table 3.1 is 55, three of which correspond to the support of non-zero codewords of size either $\nu = 6$ or $\nu = 8$, for which training is not performed.

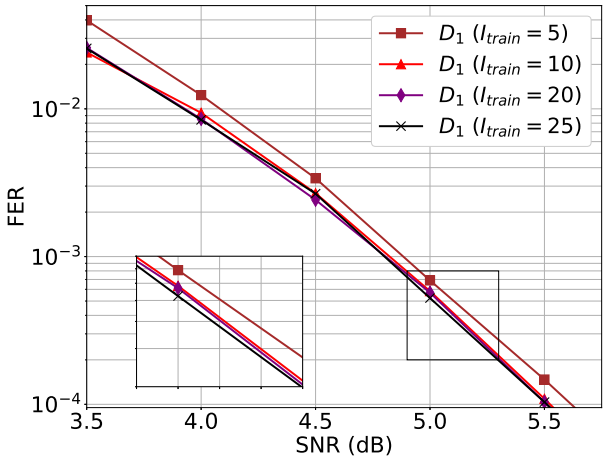
Table 3.2: Keras Parameters

Parameters	Parameters values
Optimizer (Gradient descent)	RMSprop [69] (initialized at a learning rate of 10^{-3})
Epoch number	10
Training batch size	8192
Number of batches	37 to 122 (depending on the SNR)

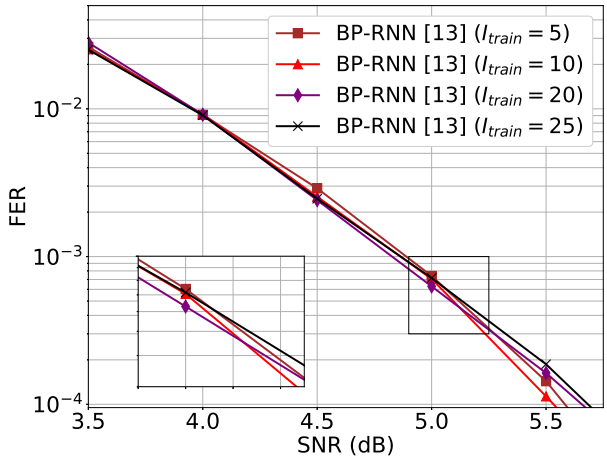
Maximum number of decoding iterations for training and testing

In this paragraph, we discuss the choice of the maximum number of decoding iterations during the training and the testing phases, denoted by I_{train} and I_{test}, respectively. (See also the discussion from [RMFS22, Section II-C].) We fix the maximum number of decoding iterations for testing our BP-RNN decoders to $I_{\text{test}} = 25$ iterations, and investigate here the impact of the maximum number of decoding iterations used at the training phase, I_{train}.

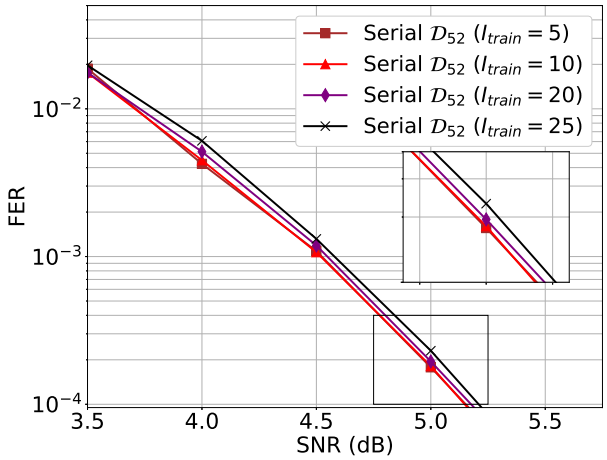
We consider the Code-1, and train all the BP-RNN decoders for $I_{\text{train}} \in \{5, 10, 20, 25\}$. Fig. 3.3 shows the frame error rate (FER) results, using $I_{\text{test}} = 25$ iterations, for (a) the BP-RNN decoder



(a) BP-RNN D_1, trained on class $7 - (5, 8, (5, 8))$



(b) BP-RNN [46]



(c) All specialized BP-RNNs, serial architecture

Figure 3.3: Impact of training parameter I_{train} on the FER performance, for BP-RNN decoders using $I_{\text{test}} = 25$ (Code-1)

specialized on the error class 7-$(5, 8, (5, 8))$, indicated as D_1 in the legend[5], (b) the unspecialized BP-RNN decoder [46], and (c) the serial architecture using the 52 specialized BP-RNN decoders. We observe no noticeable difference on the FER performance, except in Fig. 3.3a, where the FER performance for $I_{\text{train}} = 5$ is slightly degraded with respect to $I_{\text{train}} \in \{10, 20, 25\}$. In the following, we choose $I_{\text{train}} = 10$, which allows for faster training. We can note that similar observations hold for Code-2 (not shown here).

BP-RNN diversity selection

We apply now the BP-RNN diversity selection procedure from Section 3.2.3, to both Code-1 and Code-2. We proceed as follows. First, we fix SNR = 5 dB, and we train J BP-RNN decoders specialized on the J error classes ($J = 52$ for Code-1, $J = 120$ for Code-2). We order these decoders according to the procedure described in Section 3.2.3, where we use a common test set T containing 10^8 noisy codewords (SNR = 5 dB) to assess their individual error correction performance. Then we select a diversity $\mathcal{D}_Z$ of BP-RNN decoders according to (3.3), corresponding to Z different error classes. Subsequently, we only consider BP-RNN decoders specialized on the selected error classes, but we train them again for each SNR value used for testing[6].

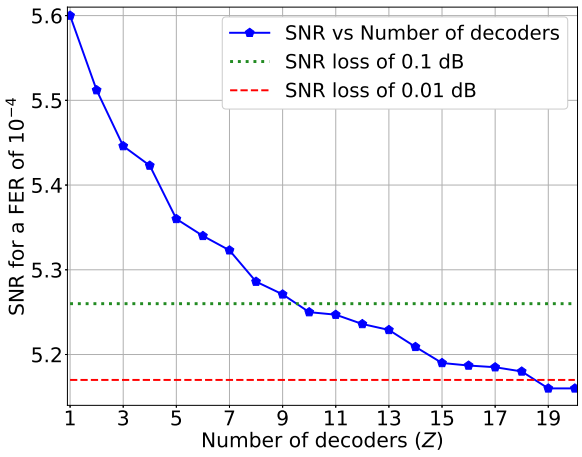


Figure 3.4: SNR for target FER = 10^{-4}, as function of Z (Code-1)

For Code-1, Fig. 3.4 shows the SNR required for achieving a target FER = 10^{-4}, as a function of the number of selected BP-RNN decoders Z (only results for $Z = 1, \ldots, 20$ are shown, since we observe no further improvement of the FER for higher Z values). We choose $Z = 10$, corresponding to an SNR loss of less than 0.1 dB, with respect to the case when all the J BP-RNN decoders are used. We use the same procedure to select a number of BP-RNN decoders for Code-2, which yields the same value $Z = 10$, for which the SNR loss is again less than 0.1 dB. Finally, in Table 3.3, we show the extended types for the $Z = 10$ error classes, corresponding to the selected decoders.

Frame error rate results

We consider the BP-RNN diversity $\mathcal{D}_{10}$ composed of the $Z = 10$ BR-RNN decoders selected in the previous section, and evaluate the FER performance, as well as the average-case complexity and decoding latency, for both parallel and serial architectures from Section 3.2.4.

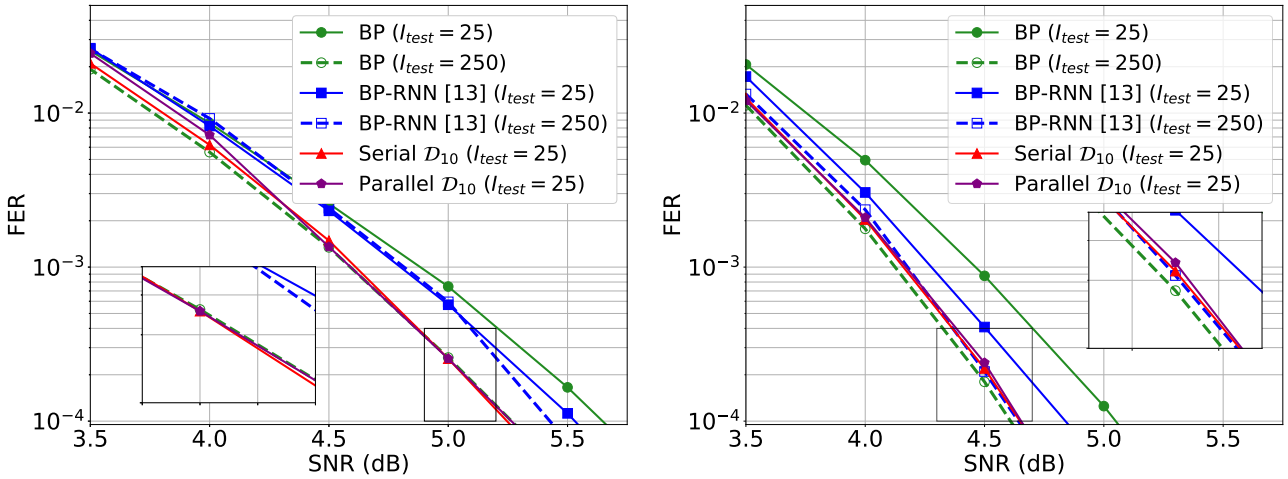
Fig. 3.5 shows the FER results for (a) Code-1, and (b) Code-2. For comparison purposes, we also show the FER performance of the BP decoder and the BP-RNN decoder from [46], with either

[5] See also Table 3.3 and the corresponding discussion from Section 3.2.5.

[6] We have also performed simulations using the BP-RNN decoders trained for SNR = 5 dB only, and testing them on different SNR values. Our simulation results were very similar to those obtained by training again the BP-RNN decoders for the actual SNR value used for testing. We chose to show simulation results for the case where the training and testing SNR values are equal, simply for consistency reasons.

Table 3.3: Extended Types (ET) for the Selected Error Classes

Code-1		Code-2	
Dec.	Error-Class (ET)	Dec.	Error-Class (ET)
D_1	7-$(5, 8, (5, 8))$	D_1	7-$(7, 11, (6, 11, 1))$
D_2	7-$(1, 9, (0, 9, 1))$	D_2	5-$(7, 9, (7, 9))$
D_3	4-$(2, 5, (2, 5))$	D_3	6-$(4, 10, (4, 10))$
D_4	7-$(3, 7, (1, 7, 2))$	D_4	7-$(5, 9, (5, 9))$
D_5	8-$(2, 9, (1, 8, 1, 1))$	D_5	6-$(8, 10, (8, 10))$
D_6	6-$(2, 7, (2, 6, 0, 1))$	D_6	7-$(5, 11, (4, 11, 1))$
D_7	5-$(1, 7, (1, 7))$	D_7	7-$(5, 8, (5, 8))$
D_8	6-$(2, 7, (1, 7, 1))$	D_8	7-$(7, 7, (7, 7))$
D_9	7-$(1, 9, (1, 8, 0, 1))$	D_9	7-$(3, 11, (3, 11))$
D_{10}	3-$(3, 3, (3, 3))$	D_{10}	7-$(7, 13, (7, 13))$



(a) FER results for Code-1

(b) FER results for Code-2

Figure 3.5: Frame Error Rate (FER) results

$I_{\text{test}} = 25$, or $I_{\text{test}} = 250$. The latter I_{test} value corresponds to the cumulative maximum number of iterations performed by the BP-RNN decoders in the diversity $\mathcal{D}_{10}$.

We observe that the parallel and serial architectures exhibit virtually the same FER performance, the corresponding curves being practically superimposed one on another. Compared to the conventional BP decoding, the BP-RNN diversity $\mathcal{D}_{10}$ produces an SNR gain of approximately 0.4 dB with respect to BP($I_{\text{test}} = 25$). This comparison is relevant to applications with strict latency requirements, since both the BP($I_{\text{test}} = 25$) and the parallel BP-RNN diversity have the same worst-case decoding latency. By way of comparison, for Code-2, a similar gain over the conventional BP has been recently reported in [23, Fig. 6], by using an automorphism ensemble decoding (AED) approach, with 16 BP decoders working in parallel[7]. If the worst-case latency constraint is relaxed, it can be observed from Fig. 3.5 that BP($I_{\text{test}} = 250$) achieves similar FER performance to the BP-RNN diversity $\mathcal{D}_{10}$. Similar considerations hold for the comparison between the (unspecialized) BP-RNN [46] and the BP-RNN diversity $\mathcal{D}_{10}$, with one noticeable exception for Code-1, for which it can be observe that increasing the number of iterations from $I_{\text{test}} = 25$ to $I_{\text{test}} = 250$ does not improve the FER performance of the BP-RNN [46] decoder (or only slightly in the low FER region). Finally, we note that using only the first decoder (D_1) of our specialized BP-RNN decoders yields similar FER performance to the unspecialized BP-RNN [46]. While this is not shown in the figure (to avoid clutter), it can be observed for

[7]We did not include the AED-16 curve from [23] in Fig. 3.5b, to avoid clutter. The gain reported in [23] was observed using 32 decoding iterations.

Code-1 by comparing the FER results in Figs. 3.3a and 3.3b.

For an evaluation of the average-case computational complexity and the average-case decoding latency we refer to [RMFS22].

3.2.6 Discussion

In this section we addressed the problem of enhancing the BP-RNN performance by exploring a decoding diversity approach, where BP-RNN decoders are specialized to specific classes of errors. Our first results revealed that the weight optimization responded differently to training sets corresponding to different error classes (Fig. 3.1), indicating an actual specialization of the decoder with respect to the error class. Then, we showed that the proposed BP-RNN diversity approach, coupled with a parallel decoding architecture, allows increasing the decoding performance without increasing the worst-case latency (Fig. 3.5). If the worst-case latency constraint is relaxed, the BP-RNN diversity performance can be attained by the conventional BP decoder, at the cost of a larger number of decoding iterations. Thus, we may conclude that the decoding grain brought by the proposed approach is essentially a gain in convergence speed, rather than a gain in error correction capability. This is actually a much more general remark that applies to most of the neural BP decoding approaches reported in the literature (and our approach is no exception): it turns out that the common characteristic of neural BP based approaches is that they increase the decoding speed rather than the error correction capability. However, the diversity decoding approach investigated in this section can be leveraged in different ways, and in the next section we further combine our approach with a reliability-based post-processing step, as a means of boosting its error correction capability.

3.3 BP-RNN Diversity with OSD Post-Processing

3.3.1 OSD decoding

OSD was first proposed in [44], as a decoding method capable to approach the ML decoding performance, for moderate-length linear block codes, with polynomial complexity. It can be used as a stand-alone decoding algorithm, exploiting the soft-output of the channel ($L_{\mathrm{ch},n}$), or as a post-processing step, exploiting the output of a soft-decision decoder ($\tilde{L}_n$).

In OSD, variable-nodes are first sorted according to their *reliability* (that is the absolute value of the corresponding soft decision). The parity-check matrix of the code is then brought to a systematic form[8], $H = [A \mid I]$, where A is a matrix of size $M \times (N-M)$ and I is the identity matrix of size $M \times M$, and so that the $K := N - M$ columns of A correspond to the most possible[9] reliable variable-nodes. By a slight abuse of language, we simply refer to variable-nodes corresponding to the columns of A as the most reliable ones, and to the remaining variable-nodes as the least-reliable ones. In OSD-0, hard-decision is made on the most reliable variable-nodes, and the least reliable ones are determined by solving the linear system given by H. Hence, decoding is successful if and only if the most reliable variable-nodes are error-free. To address the case where these variable-nodes contain errors, OSD-w considers all the possible choices of at most w errors among them. For each choice, the initial hard-decision of the corresponding variable-nodes is flipped, and the least reliable variable-nodes are determined again by solving the linear system given by H. This procedure produces a list of $\sum_{i=0}^{w} \binom{K}{i}$ codewords, from which the most likely one is selected, according to an ML rule, such as (3.4). OSD-w may closely approach the ML decoding performance, assuming the w value (referred to as OSD order) is suitably large.

[8] For simplicity, we assume here that the parity check-matrix is of rank M.

[9] Taking into account that column swaps may be needed, in case the M least reliable columns of H are not linear independent.

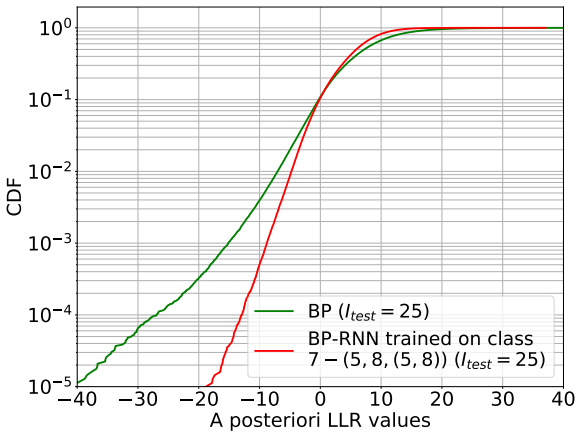


Figure 3.6: CDF of the a posteriori LLR values when decoding fails (Code-1, SNR = 4 dB, maximum number of decoding iterations $I_{\text{test}} = 25$)

3.3.2 OSD as a post-porcessing step

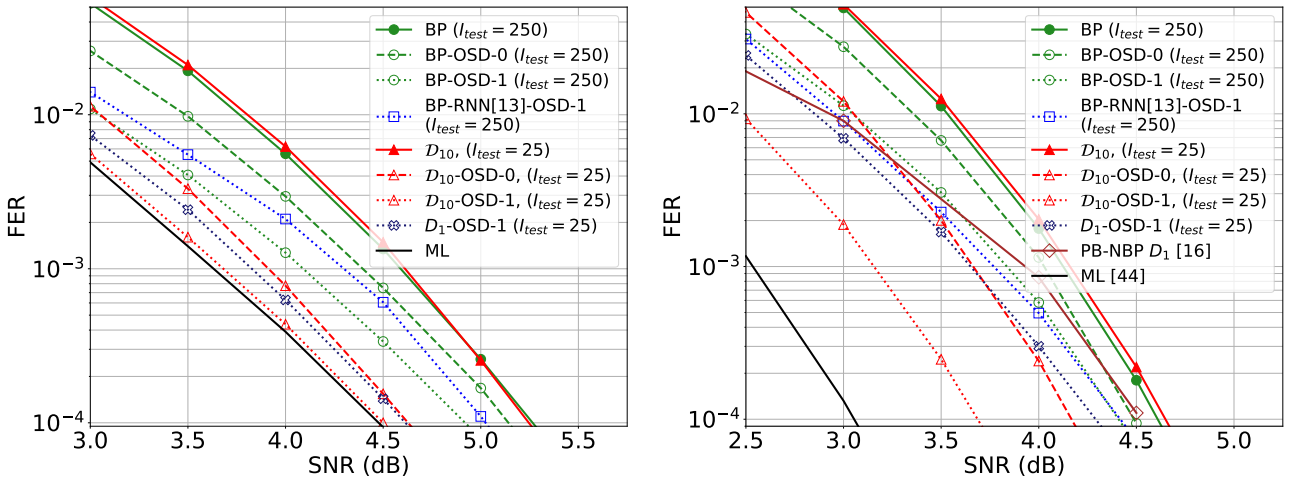
To bridge the error correction performance gap between suboptimal BP decoding and ML decoding, [29] suggested combining BP decoding with a low-order OSD ($w \leq 1$), where an OSD step is performed at the end of each iteration of the BP decoding. Here, we propose the use of OSD as a *post-processing step*, applied only in case that none of the constituent BP-RNN decoders (of the BP-RNN diversity $\mathcal{D}_Z$) outputs a codeword. In such a case, we process one OSD using the soft-decision (a posteriori LLRs) delivered by each of the constituent BP-RNN decoders. This produces a list of Z codewords (one for the OSD post-processing of each BP-RNN decoder), and the ML rule (3.4) is used to determine the most-likely one, which becomes the outputted codeword $\hat{c}$. Note that the above description applies to the both parallel and serial architectures, since OSD post-processing is only performed when all the constituent BP-RNN decoders failed to find a codeword. To reduce the complexity of the post-processing step, we also limit the order of the OSD to $w \leq 1$.

There are two main motivations behind the use of the OSD post-processing step. First, the complexity of the low-order OSD is dominated by the Gaussian elimination step, needed to bring the parity check-matrix to a systematic form. However, for LDPC codes, the sparsity of the parity-check matrix can be advantageously exploited to significantly reduce the complexity of this step. See for instance the method proposed in [70] for solving sparse linear systems, which has been adapted in [71] to derive an efficient encoding technique for LDPC codes, and in [72] to derive an efficient ML decoding algorithm for LDPC codes over erasure channels (the situation is similar for OSD, where the least reliable variable-nodes can be seen as being erased). Second, we use OSD to post-process the soft-output of BP-RNN decoders. Since the binary cross-entropy loss function used to train these decoders penalizes negative LLR values, corresponding to variable-nodes in error[10], we expect such negative LLR values to have a reduced amplitude, thus reducing the probability of error on the most reliable variable-nodes. This is illustrated in Fig. 3.6, where we plot the cumulative distribution function (CDF) of the a-posterori LLR values for the BP decoder, and for the BP-RNN decoder trained on the error class 7-$(5, 8, (5, 8))$. Moreover, we also expect OSD post-processing to benefit from the diversity brought by the use of multiple BP-RNN decoders, increasing the probability that at least one of these decoders has at most w errors among the most-reliable variable-nodes.

3.3.3 Numerical results

We consider low-order ($w = 0, 1$) OSD post-processing applied to the BP($I_{\text{test}} = 250$), the unspecialized BP-RNN($I_{\text{test}} = 250$) decoder [46], and the BP-RNN diversity $\mathcal{D}_{10}(I_{\text{test}} = 25)$. Since the parallel and the serial $\mathcal{D}_{10}$ exhibit similar FER performance, and OSD post-processing is only applied in case all

[10]Under the all-zero codeword assumption, since both the channel and the decoder are symmetric.



(a) Code-1 (b) Code-2

Figure 3.7: FER results using OSD post-processing

the BP-RNN decoders composing $\mathcal{D}_{10}$ fail to find a codeword, it follows that the OSD post-processing step yields similar performance when applied to either one of the parallel or serial architecture. We simply refer to the corresponding decoder as $\mathcal{D}_{10}$-OSD, without mention of the diversity architecture.

Simulation results are presented in Fig. 3.7 for (a) Code-1, and (b) Code-2. For Code-1, first we note that the BP-OSD-1 provides better performance than the unspecialized BP-RNN-OSD-1. Using only the first decoder of our BP-RNN diversity (D_1-OSD-1) outperforms the BP-OSD-1 by about 0.31 dB at FER $= 10^{-4}$. Using all the BP-RNN diversity ($\mathcal{D}_{10}$-OSD-1) provides an extra gain of 0.12 dB, *i.e.*, a total gain of about 0.43 dB with respect to BP-OSD-1. Furthermore, we observe that $\mathcal{D}_{10}$-OSD-1 virtually achieves the ML decoding performance, where the latter is estimated according to [44] (we also note that the OSD-3 decoder provides an accurate approximation of the ML decoding performance). Finally, a gain of 0.52 dB can be observed for $\mathcal{D}_{10}$-OSD-0 with respect to BP-OSD-0.

For Code-2, we note that the unspecialized BP-RNN-OSD-1 provides slightly better performance than the BP-OSD-1. $\mathcal{D}_{10}$-OSD-w outperforms BP-OSD-w, by 0.32 dB, for $w = 0$, and 0.72 dB, for $w = 1$, at FER $= 10^{-4}$. Using only the first decoder of our BP-RNN diversity, we observe that D_1-OSD-1 outperforms BP-OSD-1 by about 0.11 dB. For comparison purposes, we have also included in Fig. 3.7b the FER performance of the Pruning Based Neural BP (PB-NBP) decoder from [49]. Several PB-NBP variants are presented in [49], we consider here the PB-NBP decoder D_1 (see Fig. 6 in *loc. cit.*). It can be observed that $\mathcal{D}_{10}$-OSD-1 outperforms the PB-NBP decoder by 0.84 dB, our decoder achieving a FER performance at only 0.63 dB from the ML decoding.

To further analyze the diversity obtained by $\mathcal{D}_{10}$-OSD-w, we compare it with a decoding diversity combining BP and OSD, where the latter is applied periodically throughout the BP decoding iterations [29]. Precisely, we consider BP($I_{test} = 250$), and record the a-posteriori LLR values after every 25 decoding iterations. In case a codeword is not found after 250 decoding iterations (BP fails), we apply one OSD-w on each recorded set of a-posteriori values. Hence, the OSD-w is carried out ten times in case of a decoding failure, as with $\mathcal{D}_{10}$-OSD-w. The ML rule (3.4) is used afterwards to determine the final codeword.

We restrict our attention to Code-2 and, in order to further narrow the gap to ML decoding performance, we consider OSD of order $w = 1$ and $w = 2$. The corresponding FER results are shown in Fig. 3.8. We observe that the proposed $\mathcal{D}_{10}$-OSD-w performs better than BP($I_{test} = 250$) with OSD-w every 25 iterations, especially for $w = 2$, where it achieves a FER performance within 0.2 dB from the ML decoding. Moreover, it should be noticed that using the parallel decoding architecture from Section IV.B, the worst case latency of the $\mathcal{D}_{10}$ diversity corresponds to 25 decoding iterations, while the worst case latency of BP($I_{test} = 250$) is equal to 250 decoding iterations. Finally, we note that $\mathcal{D}_{10}$-OSD-2 exhibits a gain of 1 dB with respect to BP($I_{test} = 250$)-OSD-2.

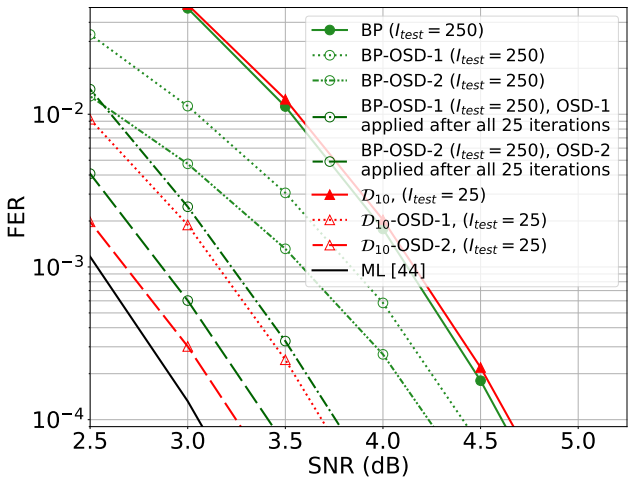


Figure 3.8: FER results using OSD post-processing of order $w = 1$ and $w = 2$ for Code-2

3.3.4 Discussion

Here, we addressed the problem of enhancing the BP-RNN performance at short code length, by exploring two complementary approaches: (1) decoding diversity, from the previous section, and (2) reliability-based post-processing. We showed that the proposed OSD post-processing step advantageously leverage the bit-error rate optimization induced by the use of the cross-entropy loss function, as well as the diversity brought by the use of multiple BP-RNN decoders. The proposed approach, combining decoding diversity and low-order OSD post-processing, provides an efficient way to bridge the gap to maximum-likelihood decoding (Fig. 3.7 and Fig. 3.8). It also opens new perspectives, as new approaches may be considered for the optimization of NN-based decoders, not to deliver the best possible bit or frame error rate performance, but merely an output that best suits the reliability-based post-processing step. Following this line of research, the next section investigates a NN-based approach to improve the OSD post-processing performance when applied to BP decoding.

3.4 Improving OSD Post-Processing for BP Decoding

As discussed in Section 3.1, we consider here the case when OSD post-processing is applied after the standard BP decoder, with the aim of proposing an alternative solution (and therefore a different performance/complexity trade-off) to the previous BP-RNN diversity based approach. For details we refer to [RMSF23].

3.4.1 Accumulated and optimized LLR for OSD post-processing

We are interested in determining an LLR accumulated during BP iterations, as suited as possible for OSD post-processing. To do so, we propose to compute the reliability of a variable-node n as a weighted sum of the a posteriori LLRs of the BP decoder, across the I decoding iterations:

$$\hat{L}_n^{(\mathrm{NS})} = \sum_{i=0}^{I} w^{(i)} \hat{L}_n^{(i)}, n \in [1, N] \tag{3.5}$$

This equation is then modeled by a simple neuron. To create the neuron training set, we first generate a set $\mathcal{T}_{\mathrm{BP}}$ of noisy codewords by considering a binary-input AWGN channel, with a BPSK alphabet (± 1) inputs, and a fixed noise variance σ^2. $\mathcal{T}_{\mathrm{BP}}$ is then decoded by the BP. (Since the AWGN channel and BP are symmetric, we may only consider the all-zero codeword.) We subsequently denote by $\mathcal{T}_{\text{BP-OSD}}$ the subset of noisy codewords of $\mathcal{T}_{\mathrm{BP}}$ for which the BP does not find a codeword after I iterations. The sets $\hat{\mathbf{L}}_\mathbf{n} := \{\hat{L}_n^{(0)}, \ldots, \hat{L}_n^{(I)}\}$, (with $n \in [1, N]$) of each noisy codeword belonging to

$\mathcal{T}_{\text{BP-OSD}}$ then constitute the training set of the neuron. This training set allows the neuron to be trained only in cases where the BP does not converge to a codeword.

To optimize the neural weights, we propose to utilize the focal loss, first introduced in [56], and defined by:

$$\text{FL}(b_n, \hat{L}_n^{(\text{NS})}) = -b_n \sigma(\hat{L}_n^{(\text{NS})})^\gamma \log\Big(1-\sigma(\hat{L}_n^{(\text{NS})})\Big) - (1-b_n)\Big(1-\sigma(\hat{L}_n^{(\text{NS})})\Big)^\gamma \log\Big(\sigma(\hat{L}_n^{(\text{NS})})\Big) \quad (3.6)$$

where b_n is the expected value of bit n, $\sigma(x) = (1+e^{-x})^{-1}$ is the sigmoid function converting the LLR value into the probability that the decoded bit is equal to zero, and $\gamma \geq 0$ is an an adjustable hyper-parameter. By assuming that the all-zero codeword is transmitted, that is $b_n = 0$ for $n \in [1, N]$, (3.6) simplifies to:

$$\text{FL}(\hat{L}_n^{(\text{NS})}) = -\Big(1-\sigma(\hat{L}_n^{(\text{NS})})\Big)^\gamma \log\Big(\sigma(\hat{L}_n^{(\text{NS})})\Big) \quad (3.7)$$

The focal loss enables to focus the training on the hardest elements to classify in the training set, by affecting them higher penalties. In our case, the most difficult vectors to classify are the $\hat{\mathbf{L}}_\mathbf{n}$ possessing $\hat{L}_n^{(i)}$ with larges amplitudes and incorrect signs, since it will result in a $\hat{L}_n^{(\text{NS})}$ with the same characteristics. Minimizing the focal loss therefore optimizes the weights to reduce the impact of such $\hat{L}_n^{(i)}$ on the computation of $\hat{L}_n^{(\text{NS})}$. Moreover, resulting $\hat{L}_n^{(\text{NS})}$ with large amplitudes and incorrect signs induce erroneous bits, which will be potentially selected among the K most reliable ones during OSD. Consequently, minimizing the focal loss reduces the probability of having erroneous bits among the K the most reliable bits. $\hat{\mathbf{L}}^{(\text{NS})}$ is thus effectively optimized for OSD post-processing. To the best of our knowledge, it is the first time the focal loss [56] is used in decoding. We noticed that training an LLR $\hat{\mathbf{L}}^{(\text{NS})}$ with a focal cost $\gamma > 0$ outperforms the training with a binary cross entropy (that is $\gamma = 0$) when OSD post-processing is applied.

3.4.2 Selecting sets of LLRs

In order to get closer to the ML decoding performance, we propose here a decoding method where an OSD post-processing is applied after each LLRs set of an ordered list $\mathcal{L}_Z$ of Z LLRs sets, with $Z \in [1, I+2]$. To construct this list, the complementarity of $\hat{\mathbf{L}}^{(\mathbf{i})} := \{\hat{L}_1^{(i)}, \ldots, \hat{L}_N^{(i)}\}$, $i \in [0, I]$, and of $\hat{\mathbf{L}}^{(\text{NS})}$ with OSD is evaluated with the number of errors remaining after OSD post-processing.

We first generate a test set $\mathcal{T}_{\text{BP-OSD}}$, according to the procedure described in the previous section. The decoding performance of the OSD post-processing is then assessed on $\mathcal{T}_{\text{BP-OSD}}$ with each $\hat{\mathbf{L}}^{(\mathbf{i})}$ and with $\hat{\mathbf{L}}^{(\text{NS})}$. We denote by $\mathcal{F}^{(i)} \subset \mathcal{T}_{\text{BP-OSD}}$ (resp. $\mathcal{F}^{(\text{NS})} \subset \mathcal{T}_{\text{BP-OSD}}$) the subset of noisy words on which $\hat{\mathbf{L}}^{(\mathbf{i})}$ (resp. $\hat{\mathbf{L}}^{(\text{NS})}$) leads to a failure during OSD-p decoding. Then, we recursively construct an ordered list of LLRs sets, noted $\mathcal{L}$. This list is initialized with $\hat{L}^{(\text{NS})}$, $\mathcal{L} = \Big\{\hat{\mathbf{L}}^{(\text{NS})}\Big\}$, since $\hat{\mathbf{L}}^{(\text{NS})}$ is optimized for OSD post-processing. To add a new LLRs set $\hat{\mathbf{L}}^{\text{new}}$ to $\mathcal{L}$, we propose to apply the following rule:

$$\hat{\mathbf{L}}^{\text{new}} = \underset{\hat{\mathbf{L}}^{(\mathbf{i})} \in \{\hat{\mathbf{L}}^{(\mathbf{0})}, \ldots, \hat{\mathbf{L}}^{(\mathbf{I})}\} \setminus \mathcal{L}}{\arg\min} \Big| \mathcal{F}_\mathcal{L} \cap \mathcal{F}^{(i)} \Big|, \quad (3.8)$$

where $\mathcal{F}_\mathcal{L} := \mathcal{F}^{(\text{NS})}$ if $\mathcal{L} = \Big\{\hat{\mathbf{L}}^{(\text{NS})}\Big\}$, $\mathcal{F}_\mathcal{L} := \mathcal{F}^{(\text{NS})} \cap \big(\cap_{\hat{\mathbf{L}}^{(\mathbf{i})} \in \mathcal{L}} \mathcal{F}^{(i)}\big)$ otherwise. The above rule is applied $I+1$ times, until $\mathcal{L}$ contains all the LLRs sets. If the minimum argument of (3.8) is not unique, an arbitrary choice is made among these values.

For $Z \leq I+2$, $\mathcal{L}_Z$ is defined as the sub-list of the Z first LLRs sets of $\mathcal{L}$. $\mathcal{L}_Z$ is thus an ordered list of LLRs sets, representing Z complementary levels of reliability with respect to OSD post-processing. As the OSD post-processing is applied to each element of $\mathcal{L}_Z$, we choose the most likely codeword among the Z candidate codewords. In the following, we note this decoding method by BP-$\mathcal{L}_Z$-OSD-p.

Finally, we point out that neither the neural modeling of the weighted sum, nor the construction of $\mathcal{L}_Z$, depend of the dimensions N and K of the considered LDPC code. As a result, the methodology presented in this paper is reproducible for any LDPC code.

3.4.3 Complexity reduction

A main practical limitation for the OSD implementation is its decoding complexity [74], since a list of $\sum_{i=0}^{p}\binom{K}{i}$ candidate codewords has to be calculated at each OSD utilisation. Hence, the OSD post-processing complexity tends to be especially costly for medium or long LDPC codes with $K > 100$ and for OSD-p with $p \geq 2$. To address this issue, the number of candidate codewords is thus usually limited by flipping only some of the most reliable variable-nodes.

As such, a procedure to compute a list of positions to be flipped during OSD-p was introduced in [74]. This list is notably determined thanks to the computation of joint error probabilities for the most reliable bits, and allows to process only the most probable candidate codewords. In [75], the most reliable bits are partitioned into segments according to reliability thresholds depending of the received noisy codeword. Only some of the segments are then selected for flipping. Both previously described methods can require a high computational cost, due in particular to the threshold computations. The authors of [76] propose for each noisy codeword to flip only the most reliable variable-nodes for which soft values do not respect amplitude thresholds.

Here, we propose a strategy to reduce the computational complexity, which does not depend of the noisy codeword. More precisely, we first put a limitation to the decoding complexity by restricting our study to OSD-2. We then reduce the complexity of OSD-2, and by extension of BP-$\mathcal{L}_Z$-OSD-2, by limiting the choices to a maximum of two errors according to their level of reliability. We thus introduce two positions thresholds, T_1 and T_2, with $T_1 < T_2$. For simplicity, the most reliable bits are also numbered from 0 to K in ascending order of reliability. For a linear code, we thus consider only up to two flips between $[0, T_2]$, with at least one flip in $[0, T_1]$ in the case of two flips. Note that the bits with sorting positions in the segment $[T_2 + 1, K]$ correspond to bits with the highest reliability, and thus with the lowest probability of error. In addition, a couple of errors in $[T_1 + 1, T_2]^2$ is less probable than a couple of errors in $[0, T_1]^2$ or in $[0, T_1] \times [0, T_2]$. Therefore, the proposed method limits the number of candidate codewords for OSD-2 by processing only error events with the highest probability of occurrence. The number of candidate codewords N_C is obtained with the following formula:

$$N_C = 1 + \binom{T_2}{1} + \binom{T_1}{2} + (T_2 - T_1)T_1 \tag{3.9}$$

The optimal choice of T_1 and T_2 is then determined by a complexity/performance evaluation thanks to (3.9) . Indeed, for a fixed number of candidate codewords N_c, all the couples satisfying (3.9) are determined. We then assessed the FER of BP-$\mathcal{L}_Z$-OSD-2 at a fixed SNR with each found couple. The couple for which the reduced complexity BP-$\mathcal{L}_Z$-OSD-2 obtains the best FER performance is thus selected.

Finally, note that for BP-$\mathcal{L}_Z$-OSD-p, the number of tested codewords when the BP fails is equal to $Z \times \sum_{i=0}^{p}\binom{K}{i}$ since Z OSD-p are performed. Therefore, the value of Z may also be limited for complexity reasons as it directly impact the number of tested codewords and the number of system resolutions.

3.4.4 Numerical results

Simulation settings

Three LDPC codes have been considered in our simulations. Their parameters are provided in Table 3.4, where $R_c := K/N$ denotes the coding rate, d_v the variable-nodes degree, and d_c the check-nodes degree.

The number of BP decoding iterations was set to 25 for the CCSDS and Tanner codes, and to 10 for the MacKay code. The OSD-p post-processing was evaluated for $p = 0, 1, 2$ in order to limit the number of tested combinations. Concerning the neuron introduced in Section 3.4.1, a set $\mathcal{T}_{\text{BP-OSD}}$ of 10000 noisy codewords was generated to create the training set. The weights, all initialized to one, were then optimized over 50 epochs, so that the focal loss converge towards a limit. Furthermore,

Table 3.4: LDPC codes parameters

	N	**K**	$\mathbf{R_c}$	$\mathbf{d_v}$	$\mathbf{d_c}$
CCSDS code [67]	128	64	0.5	3-5	8
Tanner code [77]	155	64	0.41	3	5
MacKay code [78]	1008	504	0.5	3	6

we assessed the Frame Error Rate (FER) of the OSD-p with $\hat{\mathbf{L}}^{(\mathrm{NS})}$ according to the hyper-parameter γ, and we determined empirically $\gamma = 10$ as being a good choice to penalize the misclassified LLRs. Finally, the neuron was trained for each SNR value ranging from 2.5 dB to 4.5 dB (resp. from 1.5 dB to 3.5 dB), with a step of 0.5 dB for the CCSDS code (resp. Tanner code/MacKay code). In addition, for each SNR value, a test set $\mathcal{T}_{\text{BP-OSD}}$ of 10000 noisy codewords was generated, and a list $\mathcal{L}$ was thereby constructed according to the procedure described in section 3.4.2. We compare then the different decoding strategies proposed in this paper for the three codes, in terms of FER. Reported SNR gains are evaluated at a FER of 10^{-4}.

The CCSDS code

On Fig. 3.9, the performance of CCSDS decoding for an OSD-p post processing with $\hat{\mathbf{L}}^{(\mathrm{NS})}$ is compared with the performance of an OSD-p post processing using $\hat{\mathbf{L}}^{(\mathrm{S})}$ as a reliability measure. We observe with $\hat{\mathbf{L}}^{(\mathrm{NS})}$ a slightly better performance than $\hat{\mathbf{L}}^{(\mathrm{S})}$ for $p = 1$ (similar for $p = 0$). For an order $p = 2$, $\hat{\mathbf{L}}^{(\mathrm{NS})}$ allows to obtain a gain of 0.16 dB with respect to $\hat{\mathbf{L}}^{(\mathrm{S})}$. As a result, $\hat{\mathbf{L}}^{(\mathrm{NS})}$ becomes more and more suited to OSD when the OSD order increases.

In the following, we consider a maximum budget of 3 OSD-p post-processing. To start with, the set $\mathcal{L}_3$ is determined from $\mathcal{L}$, as explained in section 3.4.2. The performance of OSD-p post-processing with $\mathcal{L}_3$ and with $\hat{\mathbf{L}}^{(\mathrm{NS})}$ alone are illustrated in Fig. 3.10. We notice that applying an OSD-p after each LLRs set of $\mathcal{L}_3$ provides an increasing gain with respect to the order p. Indeed, the gain is respectively of 0.12 dB, 0.22 dB, and 0.27 dB for $p = 0, 1, 2$. In addition, it can be observed that BP-$\mathcal{L}_3$-OSD-2 achieves a FER performance at only 0.17 dB from ML decoding [73].

Finally, OSD-p post-processing is applied after a decoding diversity of 3 BP-RNNs, denoted $\mathcal{D}_3$. The construction method of this decoding diversity is detailed in [79]. For each BP-RNN of $\mathcal{D}_3$, a single neuron is optimized with the parameters described in section 3.4.4. Three $\hat{\mathbf{L}}^{(\mathrm{NS})}$ reliability are thus computed and assessed with an OSD-p post-processing. An ML rule decides of the the final codeword. We note this diversity approach by $\mathcal{D}_3$-$\hat{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$, and the corresponding results are shown in Fig. 3.10. It can be observed that BP-$\mathcal{L}_3$-OSD-0 provides a slight improvement over $\mathcal{D}_3$-$\hat{\mathbf{L}}^{(\mathrm{NS})}$-OSD-0. The gain increases to 0.1 dB for $p = 1$, and then to 0.12 dB for $p = 2$. As a result, using the BP decoder alone and constructing a list of complementary LLRs sets is a better strategy in term of FER performance with OSD-p post-processing.

The Tanner code

The simulation results obtained with the Tanner code are presented on Fig. 3.11, with the same methodology. The decoding by BP-$\mathcal{L}_3$-OSD-0 provides a gain of 0.16 dB with respect to BP-$\hat{\mathbf{L}}^{(\mathrm{NS})}$-OSD-0. This gain remains similar when the OSD order is 1 or 2. Furthermore, we observe that BP-$\mathcal{L}_3$-OSD-2 nearly reaches the ML decoding performance.

For this code, we also consider a decoder BP-$\mathcal{L}_3$-OSD-2 operating with a complexity reduced by 50%, as described in Section 3.4.3. This complexity reduction amounts to $N_C = 3121$ instead of the BP-$\mathcal{L}_3$-OSD-2 $3 \times \sum_{i=0}^{2} \binom{64}{i} = 6243$ codewords tested at each BP decoding failure. An optimal threshold couple (T_1, T_2) for BP-$\mathcal{L}_3$-OSD-2 is thus determined for each SNR value. The corresponding performance is illustrated in Fig. 3.11. We notice that the complexity reduction of 50% induces a degradation of only 0.08 dB with respect to BP-$\mathcal{L}_3$-OSD-2 with no reduction.

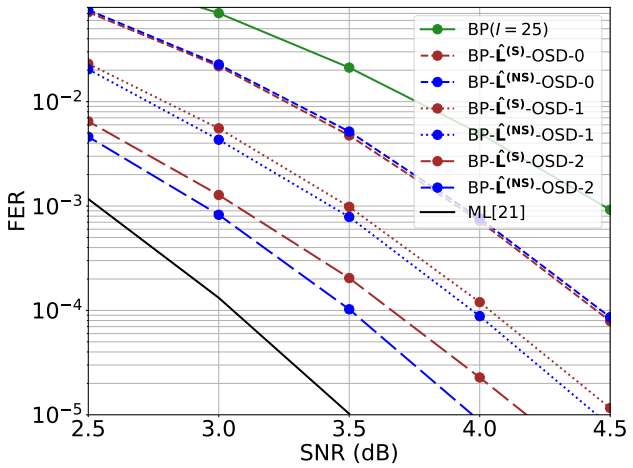


Figure 3.9: FER for CCSDS code, $\hat{L}^{(\mathrm{S})}$ vs $\hat{L}^{(\mathrm{NS})}$.

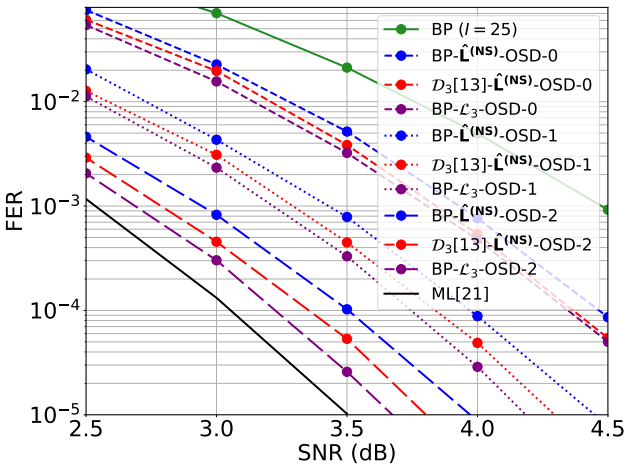


Figure 3.10: FER for CCSDS code.

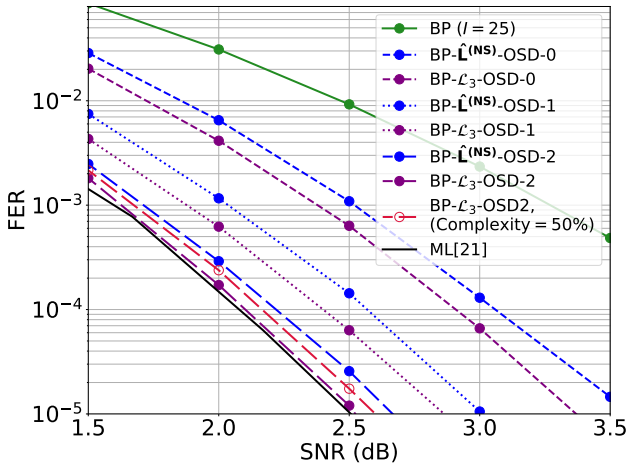


Figure 3.11: FER for Tanner code.

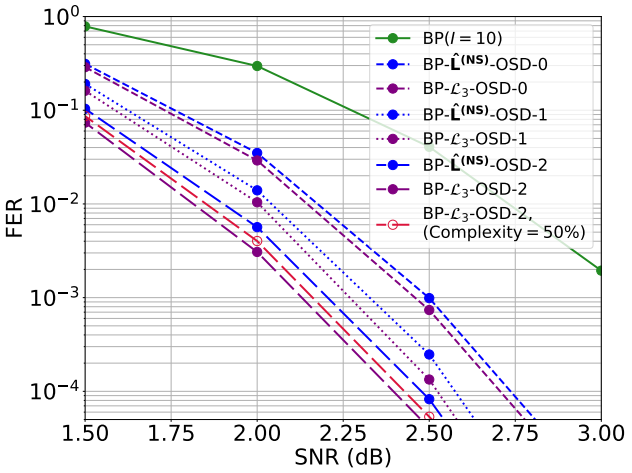


Figure 3.12: FER for MacKay code.

The MacKay code (extension to longer code)

Fig. 3.12 shows the simulations results for the MacKay code. This code possesses higher dimensions N and K than both previously discussed codes, but as explained in Section 3.4.2, the methodology is reproducible. It can be observed that BP-$\mathcal{L}_3$-OSD-p exhibits small improvements with respect to BP-$\hat{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$, up to 0.1 dB for an order $p = 2$.

However, since $K = 504$, $N_c = 381783$ candidate codewords are computed at each BP decoding failure for BP-$\mathcal{L}_3$-OSD-2. Consequently, a complexity reduction becomes mandatory for the MacKay code. As such, the decoder BP-$\mathcal{L}_3$-OSD-2 with a complexity reduction of 50% is assessed. An optimal threshold couple is hence calculated for each SNR value. We observe that the decoder BP-$\mathcal{L}_3$-OSD-2 with a complexity reduction of 50% tends to be almost as efficient than the standard BP-$\mathcal{L}_3$-OSD-2 decoder.

3.4.5 Discussion

In this section, we addressed the problem of improving the OSD post-processing performance when applied to BP decoding. To this end, the soft input of the OSD post-processing step was defined as a weighted sum of a posteriori LLRs across the BP decoding iterations, which was conveniently modeled and optimized by a simple neural approach. We also proposed the use of the focal loss function to optimize the neural weights, shown to be better suited than the binary cross-entropy loss function for this specific task. Following a diversity approach, comparable to the one proposed in Section 3.3,

we considered a multiple OSD post-processing strategy, where each OSD processes either the above weighted (neural) sum, or the a posteriori LLRs of the BP at some specific decoding iterations. Finally, we proposed a method to reduce the number of candidate codewords in the OSD post-processing step, and showed that the proposed approach is scalable for long LDPC codes and, depending on the code length, it allows approaching or covering a significant part of the gap to maximum-likelihood decoding.

4 Learned message passing receivers for multi-user MIMO communications

4.1 Motivation

This final contribution addresses multi-user MIMO (MU-MIMO) systems with multiple antennas at both transmission and reception. It is well-known that optimal multi-user detection becomes rapidly untractable due to exponential scaling in the number of active users, transmit antennas, and bits per symbol. Fortunately, the model structure induced by this type of communication can be represented as a factor graph, enabling the use of low-complexity message-passing algorithms based on variational inference [80], such as Vector-Expectation-Propagation (VEP) [81] or Approximate-Message-Passing (AMP) [82], for detecting the transmitted symbols. These algorithms are iterative and converge towards the decision of the optimal detector. In addition, being soft-input soft-output detectors by nature, they can be be used in combination with a soft-input soft-output channel decoder to further improve detection performance by exploiting the turbo principle and iterating multiple times between the approximate detector and the decoder. However the derivation of the VEP and AMP Bayesian detectors relies on specific assumptions regarding the distribution of the transmitted signals, which are not always valid in practice. In realistic transmission scenarios, these assumptions may be violated, motivating the use of neural networks and deep learning techniques. Our contribution is twofold. First, we unfold the turbo-iterations to learn certain scalar parameters for which we don't have an analytic expression. The goal is to achieve an optimal balance between a posteriori and extrinsic information in the soft messages exchanged between the Bayesian detector and the channel decoder throughout the iterative turbo process. We then demonstrate how memory can be incorporated across iterations by establishing a parallel with GRU-based models. Second, we investigate the use of neural networks on the graphs induced by communication models. A class of neural networks that naturally fits this framework is the message-passing graph neural network (MPGNN). This emerging architecture is gaining increasing attention in deep learning for physical-layer applications. We demonstrate that tailoring this architecture to the specific properties of the underlying communication model can simultaneously reduce complexity and improve performance.

Related publications. The above contributions have been published in:

[MPMC24a] A. Michon, C. Poulliat, A. Mekhiche, A. M. Cipriano, "Extrinsic Versus APP Information Feedback in Turbo VEP MU-MIMO Receivers: Optimization Via Deep Unfolding", in Proc. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seoul, Korea, Apr. 2024.

[MPC24b] A. Michon, C. Poulliat, and A. M. Cipriano, "Learning Modified Gated Recurrent Units for Information Feedback in Unfolded Turbo VEP MU-MIMO Receivers", in Proc. *IEEE International Conference on Communications (ICC)*, Denver, CO, USA, June 2024.

[MPC25b] A. Michon, C. Poulliat, and A. M. Cipriano, "Message Passing GNN for Graph Based Wireless Communication Models", in Proc. *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, Barcelona, Spain, May 2025.

Since multi-user MIMO communications were not the central project of the project, we only provide a short executive summary of each of the above contributions in the following. The interested reader is refered to the above papers for more details, results, and discussions.

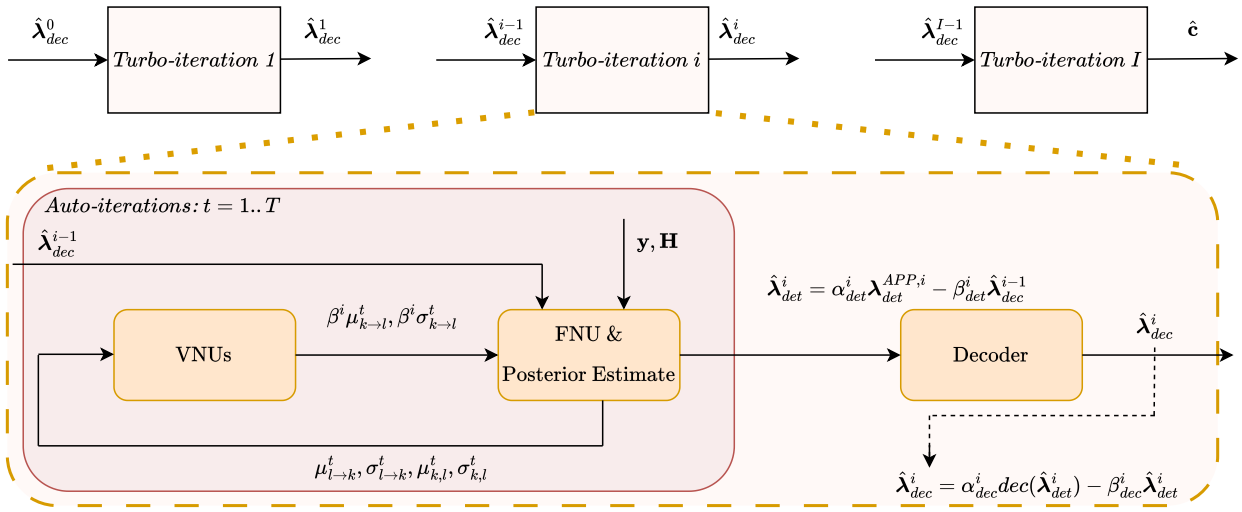


Figure 4.1: Unfolded Turbo VEP

4.2 Leveraging learning to balance extrinsic vs APP information feedback in Turbo VEP MU-MIMO receivers

Multi-User Multiple-Input Multiple-Output (MU-MIMO) communication systems with iterative detection and decoding face a fundamental challenge: determining the optimal balance between extrinsic and a posteriori (APP) information feedback in turbo receivers. This optimization problem becomes particularly complex in Vector Expectation Propagation based receivers, which operate as doubly iterative algorithms with both auto-iterations (within the detector) and turbo-iterations (between detector and decoder). We address this challenge through two distinct deep learning approaches.

4.2.1 Deep Unfolding with Learnable Scaling Factors

The first approach employs classical deep unfolding to jointly optimize hyperparameters in turbo VEP receivers. As illustrated in Fig. 4.1, the method introduces learnable scaling factors (α, β) for each turbo-iteration, transforming the information exchange according to:

$$\hat{\lambda}^{l}_{\text{det}} = \alpha_{\text{det},l}\lambda^{\text{APP},l}_{\text{det}} - \beta_{\text{det},l}\hat{\lambda}^{l-1}_{\text{dec}} \tag{4.1}$$

$$\hat{\lambda}^{l}_{\text{dec}} = \alpha_{\text{dec},l}\lambda^{\text{APP},l}_{\text{dec}} - \beta_{\text{dec},l}\hat{\lambda}^{l}_{\text{det}} \tag{4.2}$$

This parameterization allows the network to learn the optimal trade-off between extrinsic ($\alpha \to 0, \beta \to 1$) and APP ($\alpha \to 1, \beta \to 0$) information for each iteration through gradient descent optimization using Binary Cross Entropy loss.

4.2.2 Modified Gated Recurrent Unit Architecture

The second approach builds upon the previous one, but with the objective of adding introducing memory across turbo-iterations in the scaling of the soft information. The general principle is illustrated in Fig. 4.2. The main idea is to calculate the updated soft information as an exponential moving average of the new LLR and the one calculated at the previous iteration. By recognizing that such a sequential operation has much in common with the operation implemented by a standard GRU cell, we have proposed a simple, modified GRU architecture having only four scalar learnable parameters to scale soft-information exchange across turbo iterations. The resulting, hardware-efficient and numerically-optimized modified GRU cell (extGRUv2) is summarized in Table 4.1.

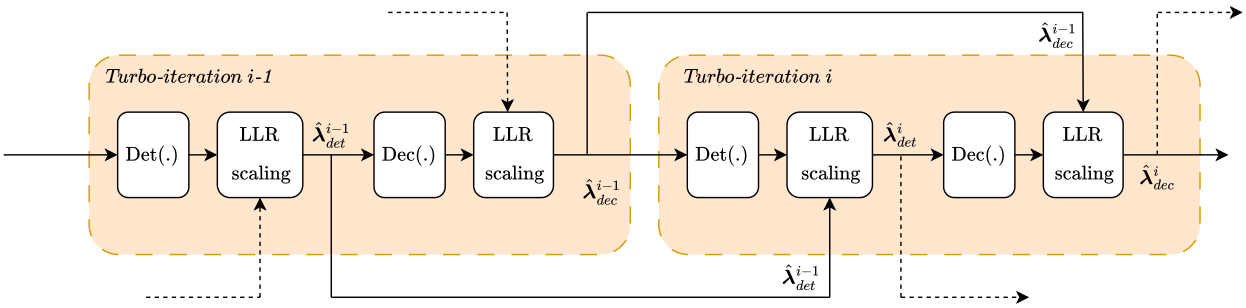


Figure 4.2: Introducing memory to scale soft information exchange across iterations

Algorithm 3: extGRU v2

Input: $\lambda^{APP,l}$, $\lambda^{apriori,l}$, h^{l-1}

$$z^l = \bar{\sigma}(\lambda^{APP,l}\phi(\lambda^{APP,l})w_z + \phi^{-1}(h^{l-1})h^{l-1}u_z)$$
$$\hat{h}^l = \phi(\lambda^{APP,l}w_h - \lambda^{apriori,l}u_h)$$
$$h = (1 - z^l)\hat{h}^l + z^l h^{l-1}$$

Table 4.1: Our extGRU v2 cell to scale soft-information (learnable parameters (w_*, u_*) are scalar)

4.2.3 Performance comparison

An experimental validation has been carried out by simulating a MU-MIMO system with $K = 4$ single-antenna users, a BS equipped with $Nr = 4$ antennas, on a Rayleigh fading channel with perfect CSI. The simulations were done using TensorFlow and Sionna. Transmission uses a gray-mapped 16-QAM and the rate-1/2 5G-NR LDPC codes from Sionna with a codeword size of 2048 bits. We use a min-sum decoder and perform 6 inner LDPC decoding iterations per turbo-iteration. The number of inner VEP iterations is set to 3, and the number of global turbo-iterations is set to 5. The results are shown in Fig. 4.3. The two main take-away conclusions are that both deep learning approaches outperform non-learned baselines, and the extGRU v2 achieves the highest performance. In particular, the GRU-based approach uniquely captures temporal dependencies between iterations, enabling adaptive damping based on convergence history and context-aware information weighting. It also exhibited improved stability in numerical computations. As for the computational complexity, deep unfolding without memory results in minimal overhead, with only 5 parameters per turbo-iteration. The extGRUv2 requires about twice more parameters but offers enhanced adaptability.

Both approaches clearly demonstrate that learned, adaptive information exchange substantially outperforms fixed strategies in MU-MIMO turbo receivers.

4.3 Message-passing Graph Neural Networks for graph-based wireless communications

We have seen that wireless MU-MIMO communication pose significant detection challenges due to their complex dependency structures. While these systems can be effectively represented as factor graphs, traditional detection algorithms face a fundamental trade-off between computational complexity and performance. First, the performance of graph-based detection algorithms varies significantly with the underlying graph structure (sparse vs. fully connected). Also, message-passing algorithms like VEP and (Vector/Orthogonal) AMP require iterative computations that scale poorly with system size. Effectively incorporating Channel State Information (CSI) and soft information from detectors into

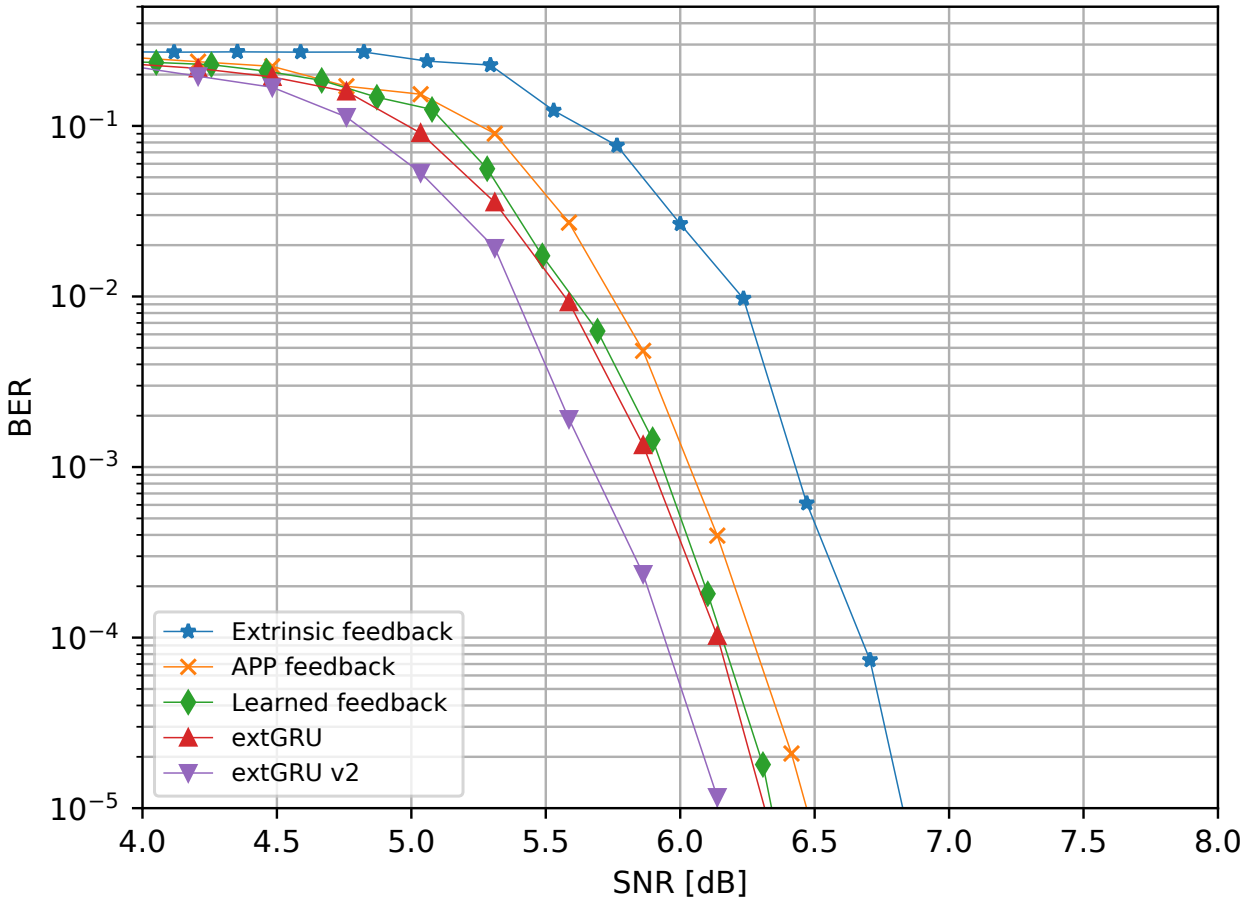


Figure 4.3: BER versus SNR for a MU-MIMO system with $K = 4$ single-antenna users, a BS equipped with $Nr = 4$ antennas, on a Rayleigh fading channel with perfect CSI. The number of inner VEP iterations is set to 3, and the number of global turbo-iterations is set to 5

neural network architectures remains non-trivial. Therefore, there is a growing interest in investigating the use of Graph Neural Networks (GNN)-based detectors to improve performance and reduce complexity. A particularly promising approach in this context is message-passing GNN [83, 84]. The motivation behind using a message-passing GNN is that both AMP and VEP approximate the joint posterior probability distribution of the transmitted symbols by the product of independent Gaussian distributions. If the variables are still correlated after the factor node update, then the hypothesis of independent Gaussian distributions does not hold, and a performance degradation results.

Different form of message-passing GNN have been proposed in the literature, building on two distinct graphical representation of the transmission model, depicted in Fig. 4.4. The first one (a) adopts a transformed bipartite graphs [85] whereas the second one (b) relies on factor graphs [86].

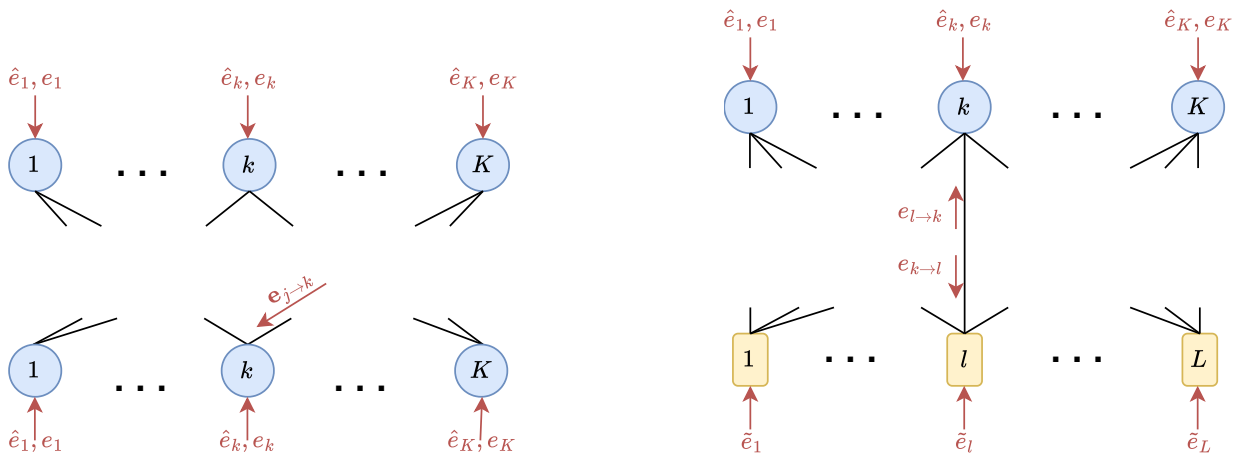
4.3.1 GEPnet: Bipartite Graph Approach

This architecture operates on a transformed graph based on $\mathbf{H}^T\mathbf{H} \in \mathbb{R}^{K\times K}$, focusing exclusively on variable node interactions (messages between users):

- **Message Passing**:

$$m_{j\to k}^{(n)} = \psi([u_k^{(n-1)}, u_j^{(n-1)}, e_{j\to k}]) \tag{4.3}$$

where $e_{j\to k} = [\mathbf{h}_k^T\mathbf{h}_j, \sigma^2]$ encodes channel correlation and noise variance.



(a) GNN module Bipartite Graph

(b) FGNN module Factor Graph

Figure 4.4: Two different graphical models of the MU-MIMO transmission

- **Node Update**: Employs Gated Recurrent Units (GRU) followed by dense layers:

$$g_k^{(n)} = \mathrm{GRU}(g_k^{(n-1)}, [\bigoplus_{j\in\mathcal{N}(k)} m_{j\to k}^{(n)}, e_k]) \tag{4.4}$$

$$u_k^{(n)} = D_u(g_k^{(n)}) \tag{4.5}$$

- **Integration with SISO Detectors**: Incorporates prior information from VEP through node attributes $e_k^t = [\mu_{\tilde{l}\to k}^t, \sigma_{\tilde{l}\to k}^t]$.

The reader is referred to [MPC25b] for the detailed description of all functions and messages involved here as well as in the next subsection.

4.3.2 FGNN: Factor Graph Neural Network

This approach directly operates on the original factor graph defined by the channel transmission matrix, with distinct variable nodes (VN) and factor nodes (FN). Message passing rules obey the following generic dual update structure:

$$m_{l\to k}^{(n)} = \psi_V([f_l^{(t,n-1)}, u_k^{(t,n-1)}, e_{l\to k}^t]) \quad \text{(FN to VN)} \tag{4.6}$$

$$m_{k\to l}^{(n)} = \psi_F([u_k^{(t,n)}, f_l^{(t,n-1)}, e_{k\to l}^t]) \quad \text{(VN to FN)} \tag{4.7}$$

FGNN requires approximately twice the GEPnet complexity due to dual node updates. We have explored different manner to modify FGNN to account for the available domain knowledge. In particular, we have proposed 3 variants using different levels of CSI:

- FGNN: No CSI, only received signal
- FGNN-CSI: Incorporates $\mathbf{H}_l\mathbf{H}_l^T$ and σ^2
- FGNN-VEP: Adds VEP-computed means and variances

4.3.3 Performance comparison

The two approaches, GEPnet and FGNN, have been compared by simulation of an uplink MU-MIMO scenario with 4 transmitting antennas and 4 antennas at the base station. The channel is an ergodic Rayleigh MIMO channel with i.i.d coefficients. Simulations were done using Tensorflow.

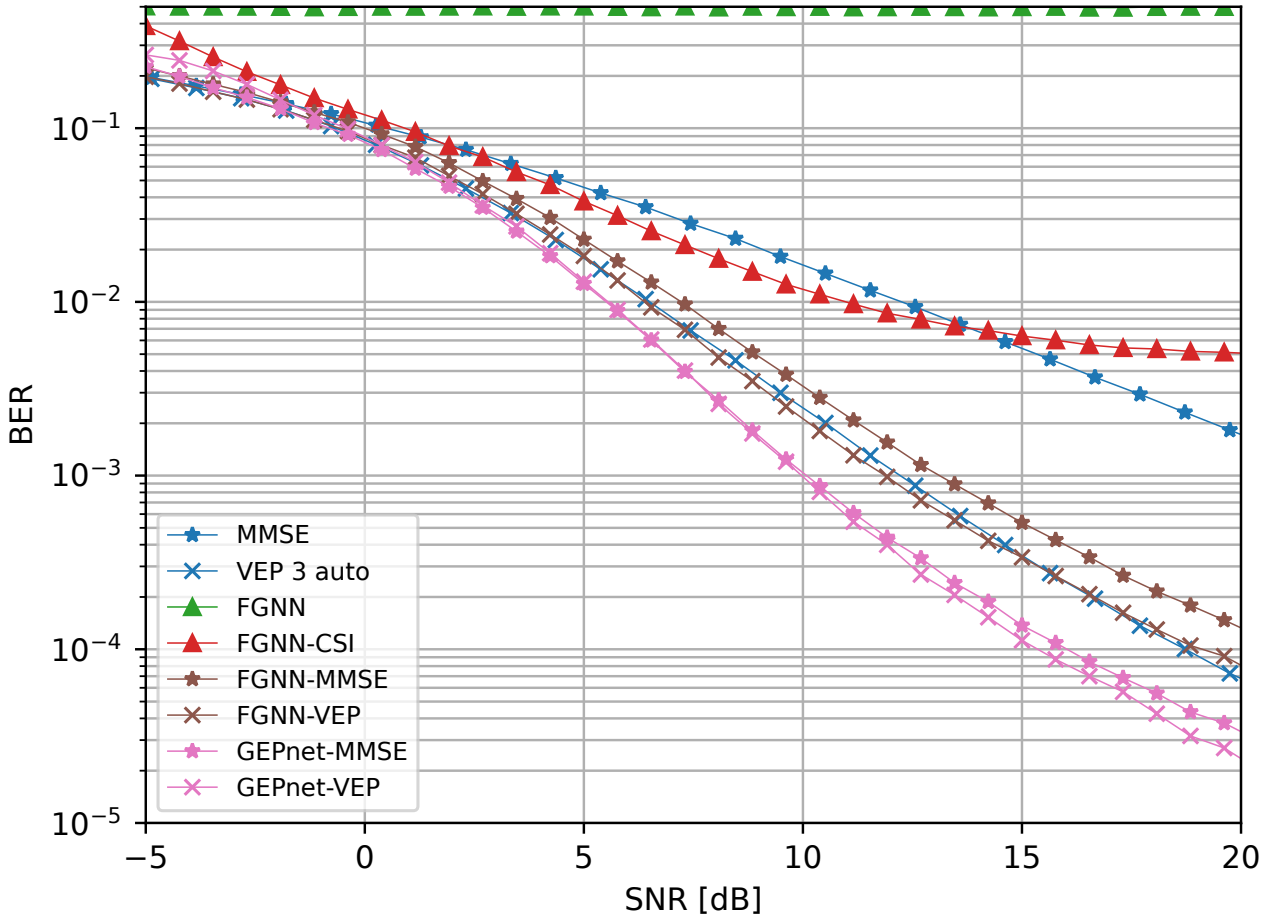


Figure 4.5: BER comparison for 4×4 MU-MIMO with Rayleigh channel and perfect CSI with QPSK modulation

The constellations are all gray-mapped QPSK. Every transmission is done with perfect CSI. We compare non-learned Minimum Mean Square Error (MMSE) and VEP to the learned modules. The results are shown in Fig. 4.5. We call "GEPnet-MMSE" the use of the GNN module with MMSE prior detection. Full lines with star markers are for MMSE and its associated learned GNN, and dotted lines with cross markers correspond to VEP. In blue are the non-learned algorithms, in brown are the ones combined with FGNN, and in pink are the ones combined with GEPnet. The curves in green are for the FGNN module and in red are the FGNN-CSI. We observe that the neural networks based on the factor graph FGNN-MMSE and FGNN-VEP is unable to outperform standard VEP. The GEPnet approach reaches better performance, especially when combined with VEP. Larger gains have been observed for larger systems, e.g. 16×16 with 16-QAM.

One can see that the FGNN performs poorly on fully connected graphs (MU-MIMO) but other simulations on ISI channels, not reported here, demonstrate that FGNN excels on sparse graphs. GEPnet demonstrates better scalability to larger MIMO systems compared to FGNN. These results highlight that the choice between factor graphs (with heterogeneous nodes) and transformed bipartite graphs (e.g., based on $\mathbf{H}^T\mathbf{H}$) fundamentally impacts detection performance with message-passing GNNs. Therefore GNN architectures must be carefully matched to the underlying communication scenario for best performance. The integration of domain knowledge through appropriate graph representations and attribute engineering can prove crucial in that respect.

5 General Conclusion

This deliverable reports on the research work carried out within WP3. Following the design space exploration performed in Deliverable D3.1, a number of possible technical contributions were identified for the remaining of the project, falling within one of the following two categories [D3.1]:

- *Developing enhanced ML-aided FEC decoders, or improving their robustness to channel uncertainty or mismatch at runtime. The main focus here is on the error-correction performance, while ensuring that complexity considerations are also properly taken into account.*
- *Developing reduced complexity decoders for non-binary codes. The main focus here is on reducing the decoding complexity, while ensuring negligible or minimal degradation of the error-correction performance.*

Of these two items, only the first one, using learning to improve binary FEC decoding, was ultimately explored, but thoroughly, and from various angles. As related to the second item, instead of developing reduced-complexity decoders for non-binary channel codes, we chose to investigate a more general form of non-binary decoding problem: using deep learning for low-complexity detection of linearly mixed modulated symbols in multi-user MIMO communications, a task akin to Euclidean-space decoding.

The first three sections of the deliverable aimed at using learning to improve the decoding of short binary codes. The very first section tackled the challenge of approaching the performance of optimal soft-decision Maximum-Likelihood Decoding (MLD) for an arbitrary linear block code, *using a model-free approach for minimum inductive bias*. Our investigation was guided by several key questions: Are current neural architectures capable of achieving near-MLD performance for codes of practical relevance? If so, how does the complexity of the model scale with parameters such as code length or minimum distance? Additionally, how does the performance of these architectures evolve as a function of the size of the training set? The focus was placed on syndrome-based neural decoding (SBND) which was found to perform best among existing model-free decoders. A performance analysis of the SBND decoder across various codes revealed two key findings: first, its frame error rate (FER) performance remains significantly inferior to that of the MLD decoder; second, current training approaches do not effectively align with the intended objective. We proposed several contributions to address these issues. These contributions are summarized below, with the key lessons learned:

1. We demonstrated that insufficient attention has been given to training data quality, especially when compared to the extensive efforts devoted to developing new model architectures. By using carefully designed fixed training datasets, we showed that simple models—such as the original GRU or the ECCT model— outperform, often by a substantial margin, the most recent architectures while requiring far fewer data samples.

2. Optimizing SBND model training enabled us to push these models to their performance limits across different codes. However, this process exposed a scalability issue: as code length increases or code rate decreases, performance plateaus. Neither expanding the dataset size nor increasing model parameters appears to bridge the gap to MLD performance.

3. Although our data-centric approach now allows us to closely approach MLD performance for high-rate, moderate-length codes, the number of parameters required remains prohibitively high as compared to traditional decoding algorithms such as Chase-2 decoding. Current SBND models lack the necessary efficiency for practical deployment.

Now that SBND model training is better understood, a promising research direction would be to identify the limitations of current architectures—with the goal of designing new, parameter-efficient models that scale more effectively.

The next two sections delved into the specific challenge of near-MLD soft-decision decoding of short LDPC codes. A fundamental observation made in Section 2 is that short LDPC codes are not inherently bad. Depending on the way they are constructed, their MLD performance can closely approach the optimal coding bounds in the short-length regime. On the other hand, BP decoding is largely suboptimal in this regime, proving unable to effectively exploit the full error correction capacity of the code. Accordingly, our contributions focused on improving the BP decoding performance of short LDPC codes, *by following a model-based approach*, and leveraging on and/or combining several techniques, such as,

- learning a set of perturbations to be applied to the input of the BP decoder whenever it fails, following a multiple-round BP (MRBP) strategy,
- exploiting a decoding diversity approach, implying several decoders working either in serial or in parallel, where each decoder is trained to decode a specific class of errors,
- complementing neural-BP or conventional BP decoding with an ordered statistics decoding (OSD) post-processing step.

Here is a summary of the main results and lessons drawn from this work:

1. Decoding approaches based solely on neural BP generally lead to an increase in the decoding speed, rather than an intrinsic improvement of the error correction capability. Our neural (BP-RNN based) decoding diversity approach is no exception to this rule. Given an worst-case latency constraint (in the form of a maximum number of decoding iterations), the BP-RNN diversity allows increasing the decoding performance, exploiting a parallel decoding architecture. However, if the worst-case latency constraint is relaxed, the BP-RNN diversity performance can be attained by the conventional BP decoder, at the cost of a larger number of decoding iterations.
2. To enhance the BP (or neural BP, *e.g.*, BP-RNN) decoding performance, we considered the use of a post-processing step, in the form of either MRBP or OSD. Although treated independently, these two approaches share some similarities, as in both cases the post-processing step is supplied with some reliability metric that needs to be learned. To some extent, one can think of the MRBB approach as a low-complexity variant of OSD, where one reuses BP as system solver, after some perturbation of its inputs. Recycling reliability metrics between MRBP or OSD approaches might be subject to future work.
3. As regard to the learned MRBP approach, we have been able to design a model that is more accurate at selecting the harmful bits to perturb than the best-known expert rules. Accordingly we have been able to reduce the number of decoding rounds required by MRBP to approach MLD performance. While this highly encouraging result demonstrates significant potential, it must be tempered by the fact that the model used to predict which bits to flip remains disproportionately complex compared to the baseline BP decoder it augments.
4. For OSD post-processing, we built upon two complementary approaches: (1) the use of decoding diversity (in the form of either multiple BP-RNN decoders, each followed by one OSDs, or multiple OSD after one single BP decoding), and (2) learning a reliability metric better suited to OSD post-processing, notably through the use of the focal loss function (instead of the usual binary cross-entropy loss function). We showed that the proposed approach, combining decoding diversity and low-order OSD post-processing, provides an efficient way to bridge the gap to MLD for short LDPC codes. While this result completely meets the expectations in terms of the error correction performance, this is not the end of the story, as the error correction performance must be balanced against the underlying decoding complexity.

The primary takeaway from these two contributions is clear: in both cases, integrating learning into the algorithm significantly reduced the number of re-encoding or re-decoding attempts required to achieve a target performance level. In both cases learning proved able to improve upon expert algorithms

or rules. This progress effectively narrows the gap between the BP decoder and the MLD decoder in a cost-effective manner. Nevertheless, the complexity of both post-processing methods remains prohibitively high compared to the inherent simplicity of the original BP decoder. As a result, the challenge of achieving near-MLD performance for short LDPC codes remains largely unresolved.

The final part of this work, detailed in Section 4, investigated the application of deep learning to multi-user detection in MIMO transmissions. Here also, the optimal detector is well known but computationally infeasible for practical implementation, necessitating the use of simplified, suboptimal approaches in radio receivers. We explored various ways to integrate learning into this context, ranging from the simplest—learning the optimal weighting between a priori and a posteriori information in the extrinsic information exchange of a turbo receiver—to more sophisticated methods, such as exchanging information via learned functions in a graph neural network. The key takeaway from this work is that deep learning can indeed help bridge the gap to optimal detector performance, but the architecture must be carefully tailored to the problem to achieve a solution with acceptable complexity.

Bibliography

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

[2] Y. Choukroun and L. Wolf, "Error correction code transformer," *arXiv preprint arXiv:2203.14966*, 2022.

[3] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for the golay codes," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 963–975, 1989.

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

[5] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing*, 2014.

[6] D. Artemasov, K. Andreev, and A. Frolov, "On a unified deep neural network decoding architecture," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, 2023, pp. 1–5.

[7] G. De Boni Rovella and M. Benammar, "Improved syndrome-based neural decoder for linear block codes," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*. IEEE, Dec. 2023.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf

[9] S.-J. Park, H.-Y. Kwak, S.-H. Kim, Y. Kim, and J.-S. No, "Crossmpt: Cross-attention message-passing transformer for error correcting codes," *arXiv preprint arXiv:2405.01033*, 2024.

[10] Y. Choukroun and L. Wolf, "A foundation model for error correction codes," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[11] Y. Choukroun, "Error correction code transformer (ecct) source code," https://github.com/yoniLc/ECCT, 2022.

[12] G. De Boni Rovella, "Solutions de décodage canal basées sur l'apprentissage automatique pour les communications de type machine-à-machine," Ph.D. dissertation, 2024, thèse de doctorat dirigée par Lacan, Jérôme et Benammar, Meryem Informatique et Télécommunications Toulouse, ISAE 2024. [Online]. Available: http://www.theses.fr/2024ESAE0065

[13] Y. Yuan, P. Scheepers, L. Tasiou, Y. C. Gültekin, F. Corradi, and A. Alvarado, "On the design and performance of machine learning based error correcting decoders," in *2025 14th International ITG Conference on Systems, Communications and Coding (SCC)*, 2025, pp. 1–6.

[14] B. Mirzasoleiman and S. Joshi, "Foundations of data-efficient learning," in *Proceedings of the Forty-first International Conference on Machine Learning (ICML)*, 2024.

[15] R. Gribonval, A. Chatalic, N. Keriven, V. Schellekens, L. Jacques, and P. Schniter, "Sketching data sets for large-scale learning: Keeping only what you need," *IEEE Signal Processing Magazine*, vol. 38, no. 5, pp. 12–27, Sept. 2021.

[16] G. Kolossov, A. Montanari, and P. Tandon, "Towards a statistical theory of data selection under weak supervision," in *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024.

[17] D. Nguyen *et al.*, "Make the most of your data: Changing the training data distribution to improve in-distribution generalization performance," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2404.17768

[18] I. Be'ery, N. Raviv, T. Raviv, and Y. Be'ery, "Active deep decoding of linear codes," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 628–640, Feb. 2020.

[19] J. Pan and W. H. Mow, "Radius domain-based importance sampling estimator for linear block codes over the awgn channel," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 1343–1348.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] E. Kavvousanos and V. Paliouras, "An iterative approach to syndrome-based deep learning decoding," in *2020 IEEE Globecom Workshops (GC Wkshps*. IEEE, 2020, pp. 1–6.

[22] J. K. S. Kamassury and D. Silva, "Iterative error decimation for syndrome-based neural network decoders," *arXiv preprint arXiv:2012.00089*, 2020.

[23] M. Geiselhart, M. Ebada, A. Elkelesh, J. Clausius, and S. Ten Brink, "Automorphism ensemble decoding of quasi-cyclic LDPC codes by breaking graph symmetries," *IEEE Communications Letters*, 2022.

[24] T. Park, S.-J. Park, H.-Y. Kwak, S.-H. Kim, and Y. Kim, "Lowering the error floor of error correction code transformer," *arXiv preprint arXiv:2502.09065*, 2025.

[25] M. Levy, Y. Choukroun, and L. Wolf, "Accelerating error correction code transformers," *arXiv preprint arXiv:2410.05911*, 2024.

[26] S.-e. Cohen, Y. Choukroun, and E. Nachmani, "Hybrid mamba-transformer decoder for error-correcting codes," *arXiv preprint arXiv:2505.17834*, 2025.

[27] M. Helmling *et al.*, "Database of Channel Codes and ML Simulation Results," http://www.rptu.de/channel-codes, 2023.

[28] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[29] M. P. Fossorier, "Iterative reliability-based decoding of low-density parity check codes," *IEEE Journal on selected Areas in Communications*, vol. 19, no. 5, pp. 908–917, 2001.

[30] H. Lee, Y.-S. Kil, M. Jang, S.-H. Kim, O.-S. Park, and G. Park, "Multi-round belief propagation decoding with impulsive perturbation for short ldpc codes," *IEEE Wireless Communications Letters*, vol. 9, no. 9, pp. 1491–1494, 2020.

[31] N. Varnica, M. P. C. Fossorier, and A. Kavcic, "Augmented belief propagation decoding of low-density parity check codes," *IEEE Trans. on Communications*, vol. 55, no. 7, pp. 1308–1317, 2007.

[32] S. Scholl, P. Schläfer, and N. Wehn, "Saturated min-sum decoding: An "afterburner" for ldpc decoder hardware," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1219–1224.

[33] B. Gadat and C. Poulliat, "Modified augmented belief propagation for general memoryless channels," in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*. IEEE, 2016, pp. 191–195.

[34] P. Kang, Y. Xie, L. Yang, C. Zheng, J. Yuan, and Y. Wei, "Enhanced quasi-maximum likelihood decoding of short ldpc codes based on saturation," in *2019 IEEE Information Theory Workshop (ITW)*. IEEE, 2019, pp. 1–5.

[35] M. Zhou and X. Gao, "Improved augmented belief propagation decoding based on oscillation," in *2008 14th Asia-Pacific Conference on Communications*. IEEE, 2008, pp. 1–4.

[36] F. Carpi, C. Häger, M. Martalò, R. Raheli, and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 922–929.

[37] H. Lee, Y.-S. Kil, M. Y. Chung, and S.-H. Kim, "Neural network aided impulsive perturbation decoding for short raptor-like ldpc codes," *IEEE Wireless Communications Letters*, vol. 11, no. 2, pp. 268–272, 2021.

[38] S. Scholl, P. Schläfer, and N. Wehn, "Saturated min-sum decoding: An "afterburner" for ldpc decoder hardware," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 1219–1224.

[39] P. Kang, Y. Xie, L. Yang, C. Zheng, J. Yuan, and Y. Wei, "Enhanced quasi-maximum likelihood decoding of short ldpc codes based on saturation," in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[40] A. Kothiyal, O. Y. Takeshita, W. Jin, and M. Fossorier, "Iterative reliability-based decoding of linear block codes with adaptive belief propagation," *IEEE Communications Letters*, vol. 9, no. 12, pp. 1067–1069, 2005.

[41] T. R. Halford and K. M. Chugg, "Random redundant soft-in soft-out decoding of linear block codes," in *IEEE International Symposium on Information Theory*, 2006, pp. 2230–2234.

[42] I. Dimnik and Y. Be'ery, "Improved random redundant iterative hdpc decoding," *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 1982–1985, 2009.

[43] T. Hehn, J. B. Huber, O. Milenkovic, and S. Laendner, "Multiple-bases belief-propagation decoding of high-density cyclic codes," *IEEE Transactions on Communications*, vol. 58, no. 1, pp. 1–8, 2010.

[44] M. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, 1995.

[45] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *IEEE Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016, pp. 341–346.

[46] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be?ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[47] X. Xiao, B. Vasić, R. Tandon, and S. Lin, "Designing finite alphabet iterative decoders of LDPC codes via recurrent quantized neural networks," *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 3963–3974, 2020.

[48] X. Xiao, N. Raveendran, B. Vasić, S. Lin, and R. Tandon, "Faid diversity via neural networks," in *2021 11th International Symposium on Topics in Coding (ISTC)*. IEEE, 2021, pp. 1–5.

[49] A. Buchberger, C. Häger, H. D. Pfister, L. Schmalen, and A. G. i Amat, "Pruning and quantizing neural belief propagation decoders," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 1957–1966, 2020.

[50] N. Doan, S. A. Hashemi, E. N. Mambou, T. Tonnellier, and W. J. Gross, "Neural belief propagation decoding of CRC-polar concatenated codes," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[51] S. Han, J. Oh, K. Oh, and J. Ha, "Deep-learning for breaking the trapping sets in low-density parity-check codes," *IEEE Transactions on Communications*, vol. 70, no. 5, pp. 2909–2923, 2022.

[52] T. Richardson, "Error floors of LDPC codes," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 3, 2003, pp. 1426–1435.

[53] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Predicting error floors of structured LDPC codes: Deterministic bounds and estimates," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 908–917, 2009.

[54] B. J. Frey, R. Kotter, and A. Vardy, "Skewness and pseudocodewords in iterative decoding," in *IEEE International Symposium on Information Theory*, 1998, p. 148.

[55] D. J. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, 2003.

[56] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[57] M. Karimi and A. H. Banihashemi, "On characterization of elementary trapping sets of variable-regular LDPC codes," *IEEE Transactions on Information Theory*, vol. 60, no. 9, pp. 5188–5203, 2014.

[58] Y. Hashemi and A. H. Banihashemi, "On characterization and efficient exhaustive search of elementary trapping sets of variable-regular LDPC codes," *IEEE Communications Letters*, vol. 19, no. 3, pp. 323–326, 2015.

[59] H. Falsafain and S. R. Mousavi, "Exhaustive enumeration of elementary trapping sets of an arbitrary Tanner graph," *IEEE Communications Letters*, vol. 20, no. 9, pp. 1713–1716, 2016.

[60] S. Abu-Surra, D. DeClercq, D. Divsalar, and W. E. Ryan, "Trapping set enumerators for specific LDPC codes," in *IEEE Information Theory and Applications Workshop (ITA)*. IEEE, 2010, pp. 1–5.

[61] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 181–201, 2010.

[62] S.-C. Wang, "Artificial neural network," in *Interdisciplinary computing in Java programming*. Springer, 2003, pp. 81–100.

[63] M. Karimi and A. H. Banihashemi, "Efficient algorithm for finding dominant trapping sets of LDPC codes," *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 6942–6958, 2012.

[64] G. B. Kyung and C.-C. Wang, "Exhaustive search for small fully absorbing sets and the corresponding low error-floor decoder," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2010, pp. 739–743.

[65] X. Zhang and F. Cai, "Efficient partial-parallel decoder architecture for quasi-cyclic nonbinary LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 402–414, 2011.

[66] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Transactions on Information Theory*, vol. 52, no. 51, pp. 386–398, 2005.

[67] "Short block length LDPC codes for TC synchronization and channel codding (CCSDS 231.1-O-1)," Consultative Committee for Space Data Systems (CCSDS), Techical Report, April 2015.

[68] D. Declercq, B. Vasic, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders – part II: Towards guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4046–4057, 2013.

[69] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[70] B. A. LaMacchia and A. M. Odlyzko, "Solving large sparse linear systems over finite fields," in *Proc. of Annual Int. Cryptology Conf. on Advances in Cryptology (CRYPT0'90)*, 1990, pp. 109–133.

[71] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 638–656, 2001.

[72] D. Burshtein and G. Miller, "Efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel," *IEEE Trans. on Information Theory*, vol. 50, no. 11, pp. 2837–2844, 2004.

[73] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of channel codes and ML simulation results," www.uni-kl.de/channel-codes, 2019.

[74] M. P. C. Fossorier and S. Lin, "Computationally efficient soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. on Inf. Theory*, vol. 42, no. 3, pp. 738–750, 1996.

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

[76] D. Wu, Y. Li, X. Guo, and Y. Sun, "Ordered statistic decoding for short polar codes," *IEEE Communications Letters*, vol. 20, no. 6, pp. 1064–1067, 2016.

[77] R. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. ISTA*. Citeseer, 2001, pp. 365–370.

[78] D. J. C. MacKay, "Encyclopedia of sparse graph codes," http://www.inference.org.uk/mackay/codes/data.html, 2008.

[79] J. Rosseel, V. Mannoni, I. Fijalkow, and V. Savin, "Decoding short LDPC codes via BP-RNN diversity and reliability-based post-processing," *IEEE Trans. on Com.*, vol. 70, no. 12, pp. 7830–7842, 2022.

[80] T. Minka, "Divergence measures and message passing. microsoft research, cambridge," UK, Tech. Rep. MSRTR-2005-173, Tech. Rep., 2005.

[81] M. Senst and G. Ascheid, "How the framework of expectation propagation yields an iterative ic-lmmse mimo receiver," in *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*. IEEE, 2011, pp. 1–6.

[82] S. Rangan, P. Schniter, and A. K. Fletcher, "Vector approximate message passing," *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6664–6684, 2019.

[83] A. Scotti, N. N. Moghadam, D. Liu, K. Gafvert, and J. Huang, "Graph neural networks for massive mimo detection," *arXiv preprint arXiv:2007.05703*, 2020.

[84] H. He, X. Yu, J. Zhang, S. Song, and K. B. Letaief, "Message passing meets graph neural networks: A new paradigm for massive mimo systems," *IEEE Transactions on Wireless Communications*, vol. 23, no. 5, pp. 4709–4723, 2023.

[85] A. Kosasih, V. Onasis, V. Miloslavskaya, W. Hardjawana, V. Andrean, and B. Vucetic, "Graph neural network aided mu-mimo detectors," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2540–2555, 2022.

[86] Z. Zhang, M. H. Dupty, F. Wu, J. Q. Shi, and W. S. Lee, "Factor graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 181, pp. 1–54, 2023.

