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 D2.2

Learning-based code and coded modulation design methods

Final Report

Editor:	Charly Poulliat (Toulouse INP)
Deliverable nature:	Public
Due date:	October 31, 2025
Delivery date:	October 23, 2025
Version:	1.0
Total number of pages:	35 pages
Keywords:	

Abstract
This report discusses the learning-based code design methods proposed to address the limitations identified in D2.1, their evaluation, and the analysis of the learning outputs. It encompasses the research work carried out in Tasks 2.2 and 2.3 of the Work Package 2 (WP2) by the project's partners. WP2 aims to investigate how learning techniques can help come up with new code design paradigms or discover new code constructions, with application to selected communication scenarios that impose challenges on FEC code design that are not fully met with existing methods.

List of Authors

Partner	Author
LAB-STICC/IMTA	Charbel Abdel-Nour (charbel.abdelnour@imt-atlantique.fr)
	Stefan Weithoffer (stefan.weithoffer@imt-atlantique.fr)
	Mathieu Léonardon (mathieu.leonardon@imt-atlantique.fr)
IMS/INPB	Christophe Jégo (christophe.jego@ims-bordeaux.fr)
	Camille Leroux (camille.leroux@ims-bordeaux.fr)
	Romain Tajan (romain.tajan@ims-bordeaux.fr)
	Afaf Alaoui (afaf.alaoui@ims-bordeaux.fr)
IRIT/INP-ENSEEIHT	Charly Poulliat (charly.poulliat@enseeiht.fr)
Lab-STICC/UBS	Emmanuel Boutillon (emmanuel.boutillon@univ-ubs.fr)

Contents

Introduction

Work Package 2 (WP2) of the AI4CODE project investigates how learning techniques can help come up with new code design paradigms or discover new code constructions with application to selected communication scenarios that place challenges on FEC code design that existing methods can address only partially at best. The updated Gantt chart of WP2 is given in Fig. 1.

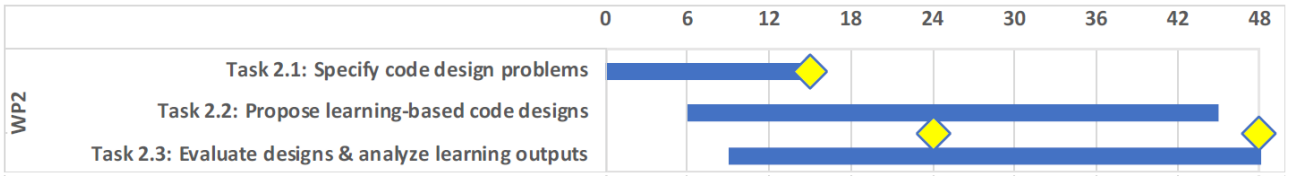

Figure 1: Gantt diagram of WP2

The first deliverable D2.1 entitled *"Specification of the code design problems"* reported activities from Task 2.1. It aimed to specify channel code design problems for which ML could be a game-changer. It provided a state-of-the-art review and gave some research directions.

This report encompasses the work carried out in **Tasks 2.2** and **Task 2.3** of the Work Package 2 (WP2) by the project's partners during the first two years of the project. Both Tasks are related to the investigation of new learning-based code or modulation design methods, the evaluation of these methods, and the analysis of the learning outputs.

The deliverable comprises three main sections, each dedicated to a particular study, and a general conclusion.

Section 1 provides a summary of the optimization efforts undertaken for the construction of good sequences for Cyclic Code Shift-Keying (CCSK) modulation, which a particular form of robust spread-spectrum waveform that is particularly appealing for IoT communication. We begin by outlining the problem to be addressed and then proceed to illustrate how machine learning gradually revealed the symmetries inherent in optimal solutions. This iterative process led us to a comprehensive understanding of the mathematical formalism, leading to the definition of the C4-sequence (a modulation scheme that has been patented).

Section 2 describes how deep reinforcement learning may be used to tackle the difficult problem of optimizing the mapping between the code symbols of a powerful non-binary code, e.g. turbo or LDPC codes, and the modulated symbols of a high-order constellation. Two different approaches are proposed and investigated: (1) an approach that optimizes non-uniform constellation for NB-codes via multi-agent reinforcement learning, (2) An approach using an optimized symbol transformation function that is added to the NB-TC encoder. In both cases, the results show significant improvement in the error floor region, compared to previous designs.

Section 3 investigates how to use supervised learning to generate polar codes with better performance under successive-cancellation list (SCL) decoding. The crux of polar code construction is to find the best frozen set for the current channel state, target code rate, and selected decoding algorithm. The proposed approach uses a two-step process in which a neural network is first trained to predict the code performance for a given frozen set in input. Then the process is reversed: the frozen set in input is optimized to minimize the predicted FER at the trained model output. Previous work already demonstrated promising performance improvement with this approach. Here the process is fine-tuned by training the neural network on an improved dataset made of frozen sets that already have good performance under SCL decoding. The goal is to push even further the performance of the new generated sequences. Then we shift our focus to designing improved polar-augmented convolutional (PAC) codes that can approach or even reach maximum-likelihood decoding (MLD) under

complexity-constrained SCL decoding. The proposed approach, which optimizes the frozen sequence based on the minimum distance and list size constraints, successfully identifies configurations that achieve near-MLD performance with a list size only half as large.

1 Learning to design sequences with good correlation properties (and rediscovering string art)

This section provides a summary of the optimization efforts undertaken for construction of CCSK sequences robust to truncation. We begin by outlining the problem to be addressed and then proceed to illustrate how machine learning gradually revealed the symmetries inherent in optimal solutions. This iterative process led us to a comprehensive understanding of the mathematical formalism, leading to the definition of the C4-sequence.

Related publications. The corresponding modulation scheme has been patented, and published in:

[B24] E. Boutillon, "Constellations Cross Circular auto-Correlation C4-sequences," in *IEEE Transactions on Communications*, vol. 72, no. 12, pp. 7664–7673, Dec. 2024.

Further on-going work on this topic and its application to industry is currently carried out within the framework of the project C4xG supported by the FRench progrAm of IP Massification for Europe in xG (**FRAME xG**).

1.1 The problem of optimizing CCSK sequences

The well known Zadoff-Chu (ZC) sequences [1], [2] are widely used in communication systems (like the 3GPP standards). They consist in a sequence of unitary complex symbols of length q that has an optimal auto-correlation function, i.e., the scalar product between the root sequence $\mathbf{z} = (z(n))_{n=0,1,\ldots,q-1}$ and any circularly rotated version $\mathbf{z}_a = z(n+a \mod q)$ is equal to q if $a = 0 \mod q$, 0 otherwise. This property guaranty that the distance between two distinct rotated ZC sequences is equal to $2q$, which make them optimally separable. This property is used in the 3GPP RACH channel (Random Access Channel) to establish a link between a base station and a device with the ALOHA protocol. However, ZC sequences do not support truncation: even with a moderate truncation lengths l, the minimum distance between rotated and truncated sequences fades very rapidly.

Our motivation was to discover new types of sequences that maintain significant distance when truncated. This property is very useful in several communication scenarios [3], [4]. The problem can be formalized mathematically. Let $\mathbf{x}$ be a complex sequence of length q and of average energy 1. Let us defined the minimum normalized square distance $D_l^2(\mathbf{x})$ between two distinct length-l truncated and rotated versions of $\mathbf{x}$, i.e.,

$$D_l^2(\mathbf{x}) = \frac{1}{l} \min_{a,b,a\neq b} \{ \left\| \mathbf{x}_a^{a+l-1} - \mathbf{x}_b^{b+l-1} \right\|^2 \}. \tag{1.1}$$

with $\mathbf{x}_a^{a+l-1} = (x(n+a \mod q)_{n=0,1,\ldots,l-1}$. The problem can be formulated as find x so that $\sum_{l=0}^{q} D_l^2(\mathbf{x})$ is maximized (the truncated sequences keep a high distance between each other).

1.2 Machine learning techniques

At the outset of our study, we had no clear direction on how to construct an optimal sequence. The possibility of surpassing the ZC sequence was even uncertain. Collaborating with our colleague Alexandru Olteanu, an expert in operational research, we turned to machine learning, more particularly genetic algorithms, to optimize the cost function associated with our problem. Promising solutions began to emerge, especially for small values of q (initially 8, then 16). Upon analyzing these solutions, we observed certain symmetries, prompting us to introduce additional constraints in the machine learning

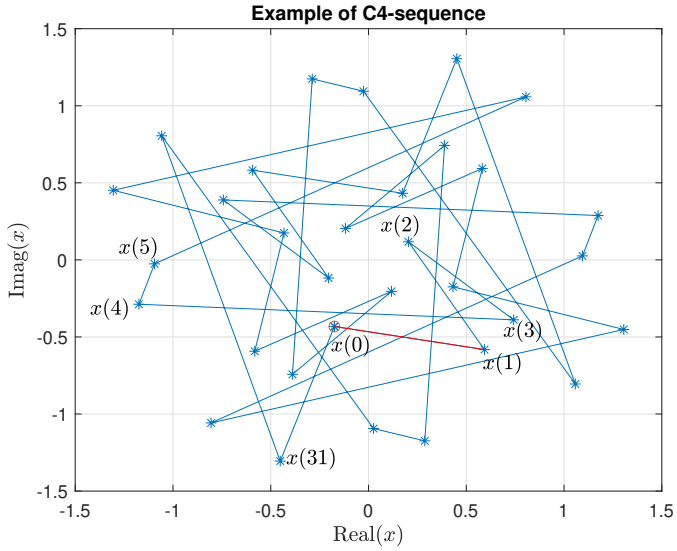

Figure 1.1: Example of a C4-sequence of size $q = 64$ generated with the seed vector $\mathbf{s} = (25, 23, 0, 11, 11, 24, 8, 22)$ (figure taken from [5]).

tools to limit the solution space accordingly. This refinement enabled us to discover optimal solutions for larger sizes ($q = 32$, then $q = 64$). Subsequently, we realized that these solutions belonged to an already documented family associated with string art patterns [4]. This insight empowered us to devise optimal solutions for any value of q. Further analysis of the mathematical properties of these optimal solutions provided the key to a more general construction of optimal sequences: the C4-sequences.

In essence, the application of machine learning techniques played a pivotal role in uncovering the solution to the broader problem of sequence construction.

1.3 Definition of C4-sequences

A constructive method to build C4-sequence is given in Algorithm 1. This result has been presented in [5].

Algorithm 1 Generation of a C4-sequence of length q by the function $\mathbf{x} = G(\mathbf{s})$

Input A seed vector $\mathbf{s}$ of size $p = q/4$ composed of $q/4$ reals on the interval $[0, q[$, a value of c in the set $\{-1, 1\}$.

Output A clockwise ($c = 1$) or anti-clokwise ($c = -1$) C4-sequence $\mathbf{x}$ of length q

for $k \leftarrow 0$ **to** $q/4 - 1$ **do**

$E_s(k) \leftarrow \sqrt{4q} \times \exp(2\pi j \frac{s(k)}{q})$

end for

$\mathbf{X} \leftarrow \text{kron}(\mathbf{E}_s, [0, \frac{1-c}{2}, 0, \frac{1+c}{2}])$

$\mathbf{x} \leftarrow \mathcal{F}^{-1}(\mathbf{X})$

Return x

Fig. 1.1 and Fig. 1.2 give two examples of C4-sequences, the first one is of size $q = 64$ which is non-unitary and the second one of size $q = 128$ where the points belong to the unit circle.

Fig. 1.3 compares the evolution of D_l^2 between a ZC sequence and a C4-sequence of Fig. 1.1. As can be seen, the C4-sequence outperforms the ZC sequence while having the same optimal property (i.e. maximum distance between two rotated versions) for $l = q$.

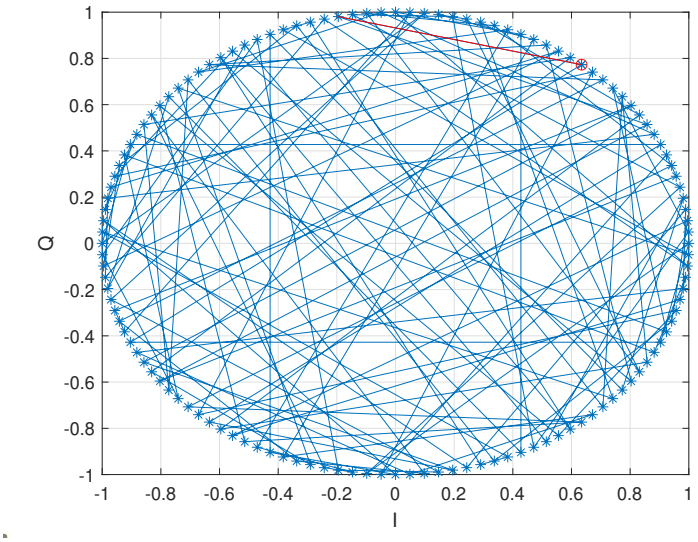

Figure 1.2: Example of unitary C4-sequence of length $q = 128$ (figure taken from [5]).

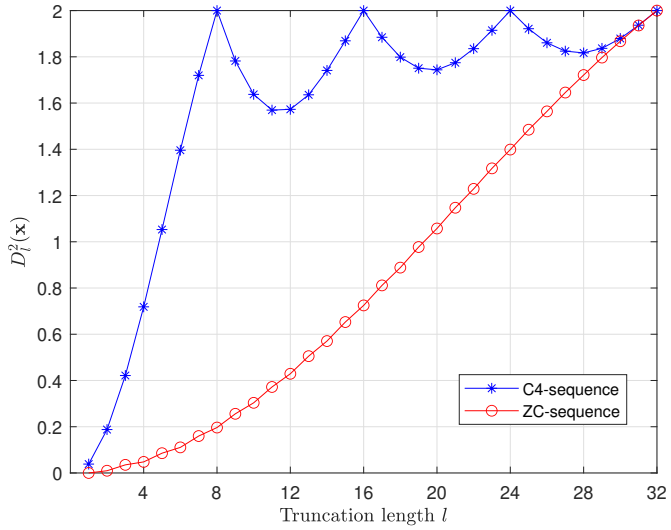

Figure 1.3: Evolution of the minimum square distance between C4-sequence $\mathbf{x}$ and a ZC sequence $\mathbf{z}$ for truncation lengths $l = 1$ to $l = q$ (figure taken from [5])

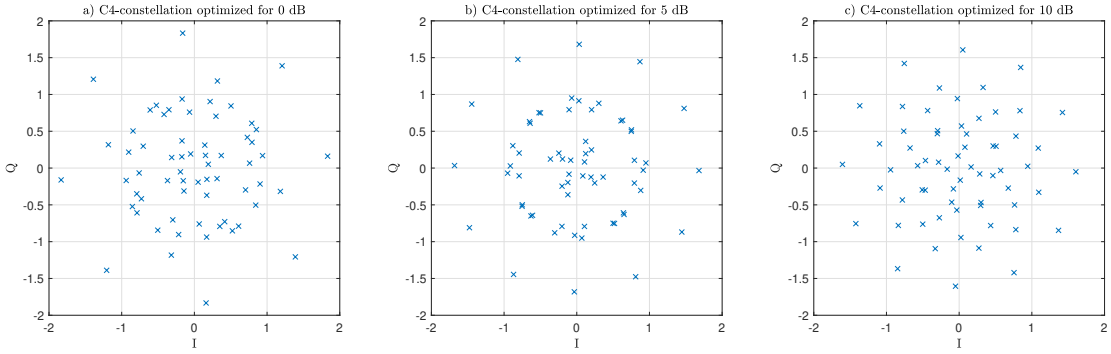

Figure 1.4: Example of optimized C4-constellations for Mutual Information (MI). For 0 dB of SNR (a) MI = 0.9998 bit/s/Hz while the channel capacity C is of 1 bit/s/Hz. For 5 dB of SNR (b), MI = 2.0536 bit/s/Hz, C = 2.057 bit/s/Hz). For 10 dB of SNR (c), MI = 3.419157 bit/s/Hz, C = 3.4594 bit/s/Hz.

1.4 Potential applications of C4-sequences

C4-sequences can have several applications in a communication system. Firstly, they can be used on their own as an alternative to Zadoff-Chu sequences. Secondly, the constellation associated with a C4-sequence can be shaped to maximize the mutual information through the AWGN channel, thereby providing geometric shaping gain (see Fig. 1.4. Finally, the concatenation of an outer non-binary code with a truncated C4-sequence as an inner code presents a very efficient and flexible communication scheme. While having a fixed outer code, the choice of truncation length provides a versatile tool for closely adapting the overall coding rate to the channel condition. It's worth mentioning that this flexibility can be effectively exploited in a hybrid-automatic request (H-ARQ) communication system. Finally, C4-sequences can be extended to create C3-sequences or C5-sequences.

2 A reinforcement learning approach to joint code and modulation design

Non-binary FEC codes reveal their potential to outperform their binary counterparts in case of a one-to-one mapping between code symbols over Galois Fields GF(q) and constellation points of the same order. As layed out in section 3.4 of deliverable D2.1 [6], this improved performance is owed to a better association between the code and the constellation. The quest for the best association should factor-in the competing sequences of symbols that constitute what we call Diverging-Converging (DC) sequences [7] and have the lowest Cumulated Euclidean Distances (CED). One way to improve the error rate performance lies in adapting the Euclidean distance between constellation points to lower the error probability of the most impacting DC sequences (i.e. having the lowest CED and/or the highest multiplicity when computing the union bound). Fully listing the DC sequences with their corresponding multiplicities is not tractable for the considered codes, it is therefore difficult to achieve such an improvement especially that sub-optimal listing techniques that aim to provide a truncated DC listing do not offer any guarantees regarding performance.

In the following we describe two approaches to further improve on this association between non-binary turbo codes (NB-TCs) and higher order constellations: (1) An approach that optimizes non-uniform constellation for NB-codes via multi-agent reinforcement learning[8], (2) An approach using an optimized symbol transformation function that is added to the NB-TC encoder. The combination of the two approaches is tackled in ongoing work.

2.1 Target code design problem and learning approach

Figure 2.1 (a) shows the structure of the one-memory element non-binary convolutional codes (NB-CCs) which serve as the component codes for the NB-TC illustrated in Figure 2.1 (b). The coefficients a_1, a_2 and a_3 in Fig. 2.1 are the GF(q) recursion and feed-forward coefficients designed to optimize the Euclidean distance spectrum conditioned by the q-QAM constellation. Note in Figure 2.1 (b), a *Symbol Transformation* Γ is additionally employed to improve performance [9]. By modifying the values of encoded symbols by one component code with respect to the other, the aim of this transformation is to avoid reproducing error-prone sequences while taking into account the effect of the interleaver. Systematic s and parity p symbols (see annotated code trellis in Fig. 2.1) are mapped to a q-QAM

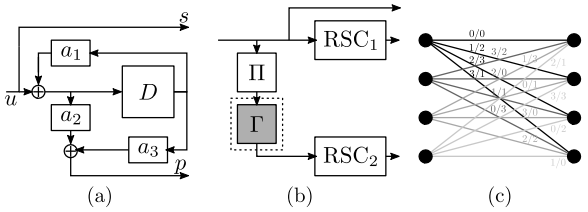

Figure 2.1: (a) NB-CC Encoder (b) NB-TC Encoder (with Transformation Γ) (c) Trellis of a GF(4) recursive NB-CC.

constellation, of the same order as the considered GF(q) for the NB code. Without loss of generality, NB-TCs over GF(64) (primitive polynomial $P_{\mathrm{GF}(64)}(D) = 1 + D^2 + D^3 + D^5 + D^6$) and 64-QAM constellations are considered in what follows. In order to achieve the best coded modulation capacity, the output symbols of the NB-TCs are to be mapped to a high-order constellation of the same order. In our case, NB-TCs over GF(64) are mapped to a 64-QAM constellation, initially respecting a gray mapping.

Following the methodology from [8], we apply the multi-agent DQN algorithm [10, 11] to try to jointly optimize (1) the positions of all constellation symbols for a non-uniform constellation, (2)

the symbol transformation function that is added to the NB-TC encoder [9] with respect to the uniform constellation and (3) The combination of the two approaches. In particular, for each of the 64 constellation points, we deploy a learning agent. We train the resulting multiple DQN agents in parallel in order to jointly optimize a single total reward called the *team reward*. In our study case over GF(64) and a 64-QAM constellation, we apply 64 DQN learning agents in parallel to optimize the minimum Euclidean distance of the NB-CC.

Using multi-agent DQN learning enables optimizing in parallel the minimum distance for each non-binary constellation symbol while considering joint effects for symbol positioning in terms of a constant total average energy per symbol and code spectrum. We recall that the minimum CED of a NB-CC can be calculated from short DC sequences as explained in [7]. Therefore, the contribution of each non-binary symbol to the minimum distance of the code can be assessed through the enumeration of all short DC sequences that have at least one of their transitions labeled by the NB symbol in question. The key problem to solve is to find a suitable reward function that is able to positively impact the most relevant short DC sequences.

Proposed reward for joint optimization of non-uniform constellation and code

In [8], we proposed a coefficient $\delta(s_i), \forall s_i \in \mathrm{GF}(q)$, that represents the contribution of the NB symbol s_i in the distance spectrum of the NB-CC:

$$\delta(s_i) = \frac{d_{\min}(s_i)}{n(d_{\min}(s_i))} \tag{2.1}$$

where $d_{\min}(s_i)$ is the minimum distance observed from short DC sequences having s_i as label of one of its transitions, and $n(d_{\min}(s_i))$ is the number of occurrences of this minimum distance. Whenever the minimum distance, $d_{\min}(s_i)$ increases or its number of occurrences $n(d_{\min}(s_i))$ decreases, the effect of the NB symbol s_i on the distance spectrum and hence performance of the NB-CC is reduced: this corresponds to large values of $\delta(s_i)$. Consequently, the value of $\delta(s_i)$ gives an insight about the impact of s_i on the performance of the code. We use $\delta(s_i)$ as the reward function for the agent in-charge of choosing the position the symbol s_i in signal space. Moreover, the final goal is to optimize a *team reward* which we define as the maximization of $\min(\delta(s_i)), \forall s_i \in \mathrm{GF}(q)$. By maximizing the minimum value of $\delta(s_i)$, we are able to optimize the spectrum of the resulting coded modulation (minimum distance and multiplicity) which represents one step further from the sole maximization of the minimum distance.

Proposed reward for optimization of the Symbol transformation

Following a similar approach as in [8], we describe a reward function for the agent which is now choosing the symbol mappings $\Gamma(S_i)$ and $\Gamma(S_j)$ for one symbol pair (S_i, S_j):

$$\delta(S_i, S_j) = \frac{d^2(S_i, S_j) + d^2(\Gamma(S_i), \Gamma(S_j)) + d_c^2(S_i, S_j)}{n(d_c)} \tag{2.2}$$

where $n(d_c)$ is the number of occurrences of this minimum distance.

2.2 Learning results

For the evaluation of the learning approaches in Section 2.1, we start from two NB-TCs as listed in Table 2.1. Note, that the codes C_1 and C_2 are the best and worst codes in terms of their CED truncated spectrum with the two first terms of the squared Euclidean distance d_1^2 and d_2^2 and the corresponding multiplicities $n(d_1)$ and $n(d_2)$ for a uniform 64-QAM constellation obtained by the classical method from [9]. In fact, the performance of any other code would lie between the ones obtained by C_1 and C_2, hence bounding the achievable performance. These codes, together with a symbol transformation obtained by the methods outlined in [9] serve as a baseline to evaluate our learning approach.

Table 2.1: Best and worst obtained codes over GF(64), with the two first terms of the squared Euclidean distance spectra d_1^2 and d_2^2 and the corresponding multiplicities $n(d_1)$ and $n(d_2)$.

Code	C_1	C_2
(a_1, a_2, a_3)	$(41, 2, 0)$	$(31, 5, 18)$
d_1^2 (d_{min}^2)	0.38	1.52
$n(d_1)$	238422	652698
d_2^2	0.57	1.61
$n(d_2)$	230886	1084014

2.2.1 Joint optimization of non-uniform constellation and code

In the case of the NB-TC using C_1 from Table 2.1 as a constituent code, Almost Regular Permutation (ARP) interleaver is used as a permutation function respecting the constraints of [12, 13]. Fig. 2.2a shows a modified constellation that respects the mapping of a 64-QAM while maximizing the minimum distance of the NB-CC C_1 of Table 2.1. Figures 2.2b and 2.2c show the impact of each NB symbol in the error prone sequences of the code computed according to Eq. (2.1). The value of $1 - \delta(s_i)$ is computed, normalized for each NB symbol s_i and associated with a color. The larger the impact of s_i on the spectrum of the code, the warmer its color and vice versa generating a heat map. Thanks to the application of the proposed method, we can observe from Fig. 2.2c that for the obtained non-uniform constellation, the effect of the majority of the constellation symbols is reduced by increasing the minimum distance in which they participate and/or by reducing their occurrence in error-prone sequences. Consequently, a mapping of the considered NB code symbols to the modified constellation of Fig. 2.2a can be expected to lead to an improved error correcting performance.

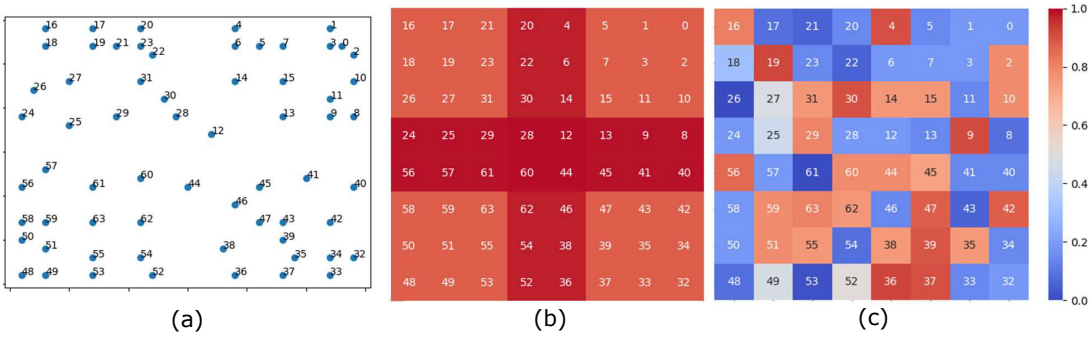

Figure 2.2: (a) Optimized symbol position for 64-QAM constellation and code C_1. (b) and (c) Heat-maps representing the contribution of each NB symbol in the minimum distance of the code for a conventional 64-QAM and for the modified 64-QAM, respectively.

A comparison between the error correcting performance of the resulting NB-TC mapped to the conventional 64-QAM constellation and to the modified 64-QAM (a) is depicted in Fig. 2.3. Indeed, applying the modified constellation, the error floor region of the considered NB-TC is lowered by about one decade, achieving a Frame Error Rate (FER) of 10^{-7}.

2.2.2 Optimization of the Symbol transformation

We evaluate the symbol transformation obtained in Task 2.2 for code C_1 and compare it against the baseline in Fig. 2.3 (a). We recall that C_1 being the worst obtained code for GF(64) illustrates the power of the symbol transformation Γ, which allows to lower the error floor by already 2 decades for the baseline case. However, our initial results show, that another decade of improvement can be obtained by employing the learning methods used in Task 2.2.

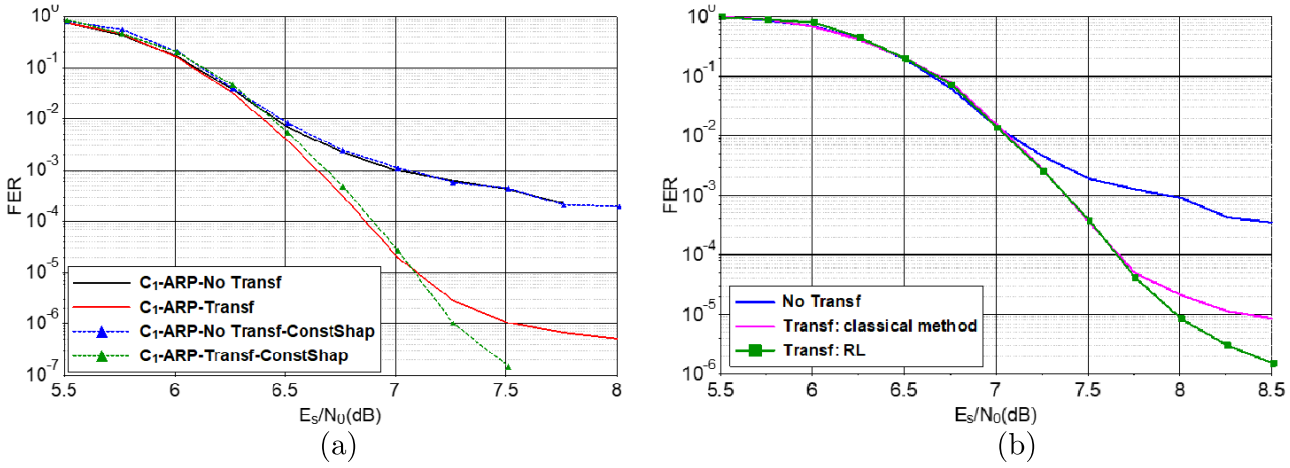

Figure 2.3: FER performance of the code C_1 with 64-QAM and Symbol Transformation, (a) With Classical Transformation and Shaping (b) With Optimized Transformation

2.3 Conclusion

In this work, we have proposed a new approach to optimize non-uniform constellations for NB codes, targeting better coded modulation performance. In particular, deep Q-learning algorithms are chosen to solve the intractable problem of enumerating all possible candidate high-order constellations. On the example of NB-TCs, we show that designed non-uniform constellations improve the error correcting performance asymptotically. This proof-of-concept clearly illustrates how ML techniques can provide solutions to complex code design tasks, which motivates to consider additional/alternative design parameters that can target not only asymptotical performance improvement, but also extend to hardware-related constraints. To complement these promising results, jointly optimizing the code construction and the positions of constellation symbols through DQN algorithms may represent a significant milestone.

3 Polar Code design tailored to SCL decoding

Introduction

Polar code [14] construction is a core process to improve the error-correction performance. This involves the splitting of an input vector into two subsets : the information set, conveying the data, and the frozen set, consisting of a predetermined values usually set to zero. The intricate challenge in polar code design lies in accurately discerning, based on the encoding rate and the code length, whether a specific position is better suited for a frozen bit or an information bit. This is fundamental to optimizing error-correction performance, tailored to a specific transmission channel and a given decoding algorithm.

Traditional polar code construction techniques exploited the subchannel polarization phenomenon to rank bit channels by reliability. This strategy, primarily designed for successive-cancellation (SC) decoding, prioritized the selection of the most reliable channels for information transmission. However, the reliance on reliability-based polar code construction did not guarantee optimal error-correction performance for decoding algorithms apart from SC. Recent advances in polar code construction include applying machine learning methods. In [15], a sequential method for SC list (SCL) decoding was introduced, formalizing the process as a maze game optimized through reinforcement learning. The game-based constructions were able to outperform the SC standard constructions of [16], for long codes, under both SC and SCL decoding. [17] and [18] proposed genetic algorithm based constructions which perform better than the state of the art codes in [19], [20] and [21] under SCL decoding. In [18] a focus was made on the SC-based decoders, namely SCL, while [17] evaluates the BP decoding as well. Apart from decoder types, the major difference between both works lies in inputs to each step of the genetic algorithm. The genetic population in [18] was randomly initialized whereas [17] adopts prior constructions based on the Bhattacharyya parameter [14] and RM-polar codes [20]. Authors in [22] and [18], have placed attention on nested polar codes which meet communication system requirements regarding memory storage. The construction problem has been modeled as a Markov decision process. The obtained codes achieve a comparable, even better, performance compared to the ones of [16] and [19]. In [23], a neural network architecture was built in order to predict the error-correction performance of polar codes given their frozen bit positions. The trained network was further used to generate more frozen bit sequences. [24] proposed a frozen set design along with dynamic frozen bit expression design. The frozen set was compactly specified using three integers. Results have shown that the compactly-specified polar codes with dynamic frozen bits outperform the state-of-the-art polar code constructions under SCL decoding.

[25] adopted a straightforward neural network architecture that have demonstrated very good performance. In this report, we will first introduce the work done in [25], explain their methodology and present their main results. Then, we will construct a training database using the method in [24] aiming to further enhance the performance of the new generated sequences.

3.1 Deep Neural Network (DNN) to predict polar code performance

In this section, the main results of [25] will be presented. The idea behind the article consists in training a neural network to predict the FER of polar code constructions at a given SNR. Subsequent sections will provide an overview of the neural network architecture and the corresponding training database. Then the Projected Gradient Descent (PGD) methodology used to generate new polar constructions will be introduced. Finally, the main results and contributions will be presented and discussed.

3.1.1 A Multilayer Perceptron (MLP) network

The MLP is a fully connected network represented by composing together many perceptrons, also called neurons, in a so called layer. A perceptron computes a linear function of an input followed by a none linearity, referred to as an activation function.

$$n(\mathbf{x}) = \sigma\left(\mathbf{w}^{\mathbf{T}}\mathbf{x} + b\right) = \sigma\left(\sum_i w_i x_i + b\right) \tag{3.1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ and b are the trainable weights, and σ is the activation function.

Thus, one layer computation can be described as the multiplication of the weight matrix $\mathbf{W}$, whose rows correspond to the trainable weights of one single neuron, by the input vector $\mathbf{x}$ (Equation (3.2)). Consequently, the MLP architecture is fully characterized by the composition of the functions computed by its layers (Equation (3.3)). Figure 3.1 illustrates the MLP network with l hidden layers, which determine the depth of the architecture, the dimension of each of them h_i is referred to as the width.

$$n_1(\mathbf{x}) = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right) = \sigma\left[\begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & & & \\ w_{h_1,1} & w_{h_1,2} & \dots & w_{h_1,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{h_1} \end{pmatrix}\right] \tag{3.2}$$

$$\mathbf{y} = n_t(\mathbf{x}) = n_l(n_{l-1}(\dots(n_1(\mathbf{x})))) \tag{3.3}$$

Where $\mathbf{x}$ is the network input and $\mathbf{y}$ its output.

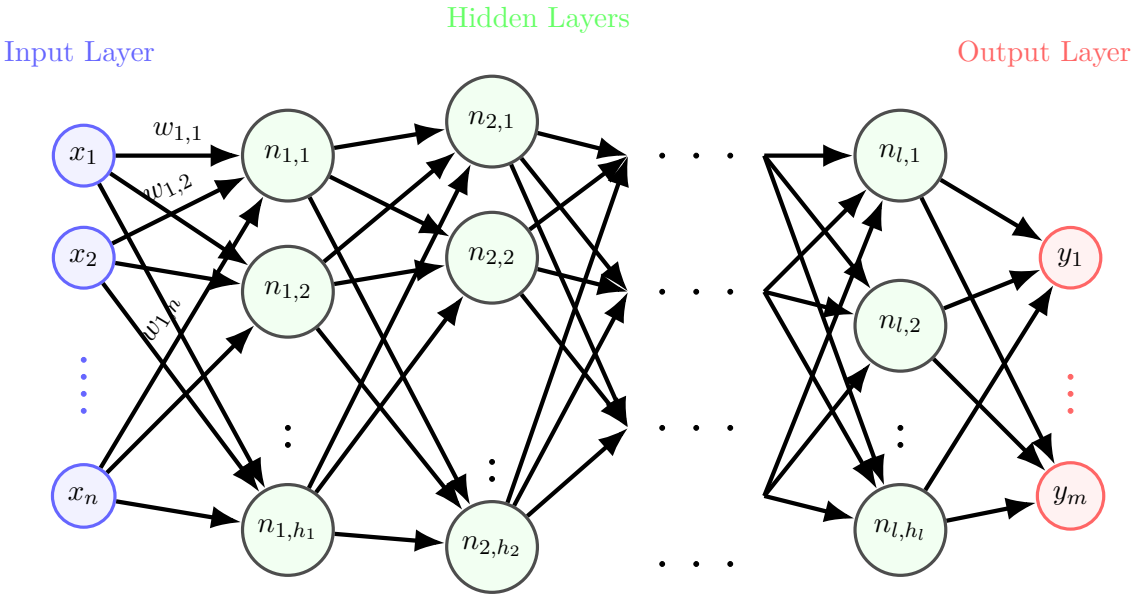

Figure 3.1: Multilayer perceptron network

Shortcuts have been introduced in the residual neural network (ResNet) architectures. They were intended to address the vanishing gradient problem in very deep networks and have demonstrated very good results, especially that they do not present additional computational complexity. They consist in adding one of the previous outputs to the input of one layer, when the two tensors have the same dimensions (Figure 3.2) .

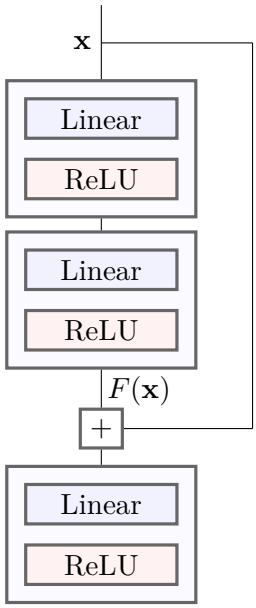

Figure 3.2: Shortcuts

3.1.2 Network training and testing

A MLP network architecture with shortcuts was adopted in [25]. The training dataset consists of pairs of frozen bit sequences $\mathbf{f}$ and their corresponding FERs in logscale at a fixed SNR . A frozen bit sequence is a binary vector whose non-zero entries correspond to the frozen positions. The sequences were obtained from Gaussian Approximation (GA) sequence $\mathbf{p}$ that sorts the channels in reliability order. A subset within the middle positions of $\mathbf{p}$, whose length was chosen empirically and set to 112, was randomly shuffled to get as many frozen bit sequences as needed (Equation (3.4)).

$$p_i^{\pi} = \pi(p_i) \qquad \forall K - r \leq i < K + r \tag{3.4}$$

where K is the block length and $r = 56$ is the shuffling range.

The sequences were generated for code length $N = 1024$ and coding rate $r = \frac{1}{2}$ at a fixed $SNR =$ 2.7dB. Their corresponding FERs were obtained by Monte Carlo simulations using a SCL decoder whose list size $L = 32$. A total of 15862 pairs were generated, 80% of which were used for training the network. Only the 112 shuffled positions were fed to the network, the remaining positions being constant all over the database. Note that both the dataset inputs and targets were standardized, which means centered around the mean and reduced to a unit standard deviation. Figure 3.3 summarizes the dataset generation methodology.

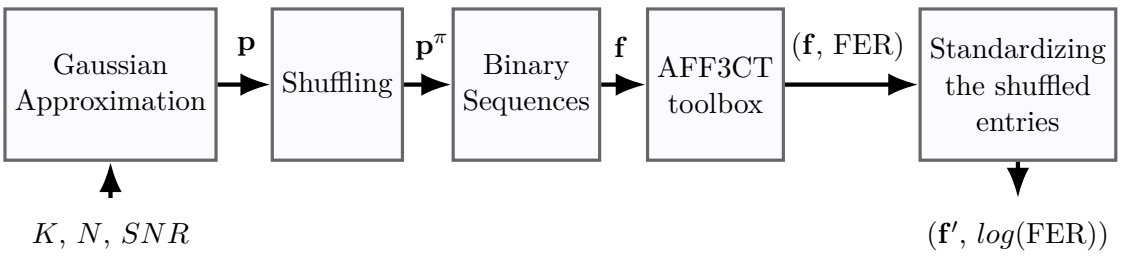

Figure 3.3: Dataset generation

As stated, the network input and output dimensions are respectively $n = 112$ and $m = 1$. The architectural parameters, depth, and width, were determined through empirical experimentations. Specifically, the depth of the network was set to $l = 3$. The width, the number of neurons in each

layer denoted as h, was set to $h = 640$ (Figure 3.1). An architecture without shortcuts has demonstrated better performance. Training was done using ADAM gradient descent algorithm to minimize the Mean Square Error (MSE) loss function between the output of the network and the FERs in logscale. Figure 3.4 represents the training and validation losses throughout epochs. The model achieves a good performance on the training set, but it seems that it does not generalize well on the validation data, which constitutes 10% of the dataset.

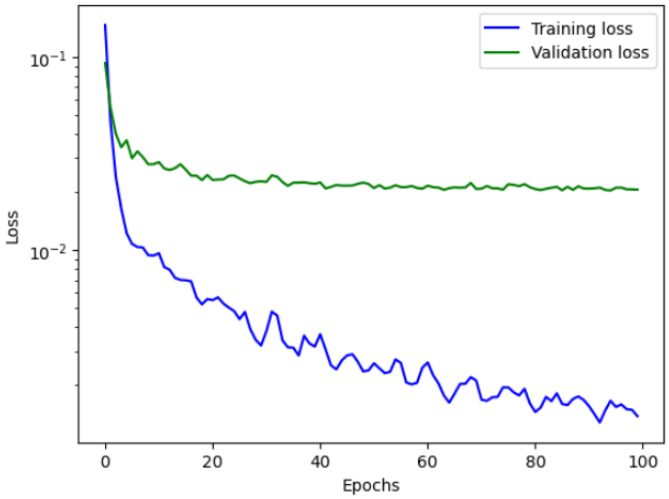

Figure 3.4: Training and validation losses throughout epochs

In order to test the performance of the network on data not previously encountered (the remaining 10% of the dataset), [25] introduces the Inflation of Error (IOE) metric as defined in Equation (3.5). The testing results are summarized in Table3.1. In average, the IOE between the estimated and the predicted FERs is no more than 5.5%, which is smaller compared to the margin of error in Monte Carlo simulations used for FER estimation.

$$IOE(\mathbf{f}, n_t) = \max\left\{\frac{\mathbf{FER_f}}{\exp n_t(\mathbf{f})}, \frac{\exp n_t(\mathbf{f})}{\mathbf{FER_f}}\right\} - 1 \tag{3.5}$$

Test loss	Average IOE	Worst IOE
0.0215	0.0541	0.2705

Table 3.1: Test results

3.1.3 Projected Gradient Descent (PGD) method

Projected gradient descent (PGD) method has been widely used to design inputs intending to induce the machine learning models to make erroneous predictions. This so called adversarial examples are corrupted versions of valid inputs, where the corruption is done by adding a small gradient-based perturbation. This method was adapted in [25] to propose new frozen bit sequences.

After training the model and fixing all its weights, PGD method is applied to produce new inputs that minimize the prediction of the network (FER).

Algorithm 2 PGD algorithm to generate new frozen sequences

$\mathbf{f} \leftarrow$ frozen sequence random initialization
for a number of iterations I **do**
  $\tilde{\mathbf{f}} \leftarrow$ quantized version of $\mathbf{f}$
  $y = n_t(\tilde{\mathbf{f}})$
  $\mathbf{f} \leftarrow \mathbf{f} - \nabla y(\tilde{\mathbf{f}})$
end for
return $\tilde{\mathbf{f}}$

3.1.4 Results

Algorithm 2 was implemented using a number of iterations $I = 5000$ and a training step $\mu = 0.1$. The resulting frozen bit sequence achieves a low FER of 10^{-5} at $SNR = 2.7$dB which is notably lower than the minimum observed within the training dataset.

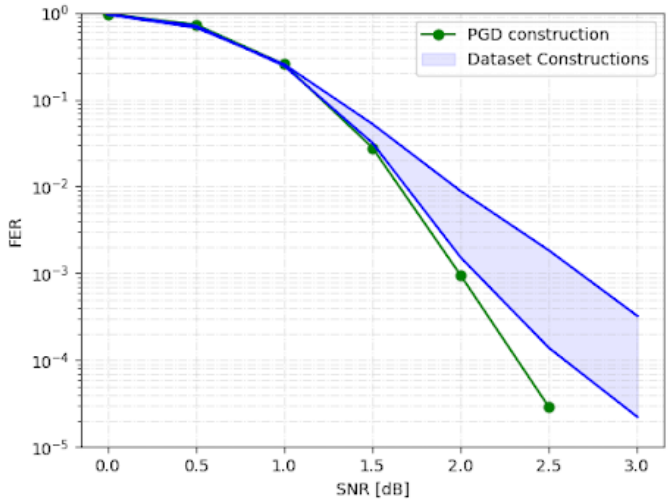

Figure 3.5: PGD construction FER vs SNR

Figure 3.5 represents the FER performance of the dataset sequences and the one generated using PGD method. The PGD sequence outperforms remarkably the dataset constructions in the high SNR region. It seems that the method has converged to a sequence of frozen bits whose minimum distance is better than the sequences of the dataset.

To conclude, the selected network architecture was initially trained to predict the FER for a given input frozen bit sequence. Subsequently, this trained network was employed to generate improved sequences whose FER performance surpass those of the training dataset. The objective of the following sections is to further improve the output of the PGD method in terms of FER.

3.2 Enhance the DNN newfound sequences

In order to generate more efficient sequences, the neural network needs to be trained using a distinct database whose sequences are slightly better than the initial ones. For this purpose, we adopt the methodology introduced in [24]. It consists in constructing polar subcodes of the Reed-Muller (RM) codes which are characterized by three integers. In the following sections, we will first provide an overview of the dataset construction methodology. Then, the training results and PGD performances will be presented.

3.2.1 The training dataset

The RM subcodes construction in [24] takes into consideration the weight distribution given a reliability sequence $\mathbf{r}$ and a minimum distance $d_{min} = 2^l$. The term weight refers to the Hamming weight of the binary expansion associated with a given integer index. The frozen set consists in the indices of weight below l, in addition to the ones of weight l and $l+1$ that are tuned according to a triplet $s = (s_0, s_1, s_2)$. s_1 and s_2 are relative values that control the number of frozen bits whose weights are l and $l+1$, respectively. s_0 is a positive integer that determines the number of the highest indices of weight l. Algorithms 3 and 4 present in detail the frozen set $\mathbf{F}$ construction methodology for a code (n, k).

Algorithm 3 triplet-tuned frozen set construction

$\mathbf{F} \leftarrow \{i \mid 0 \leq i < n, \text{weight}(i) < l\}$ ▷ freeze all indices of weight below l
$f \leftarrow n - k - s_0$
if $|\mathbf{F}| > f$ **then return** Failure
end if
$\mathbf{F} \leftarrow FreezeWeight(n, l, s_1, f, \mathbf{r}, \mathbf{F})$ ▷ freeze indices of weight l
$\mathbf{F} \leftarrow FreezeWeight(n, l+1, s_2, f, \mathbf{r}, \mathbf{F})$ ▷ freeze indices of weight $l+1$
$\mathbf{F} \leftarrow FreezeWeights(n, l+2, f, \mathbf{r}, \mathbf{F})$ ▷ freeze indices of weight $> l+2$ so that $|\mathbf{F}| = f$
for s_0 times **do** ▷ freeze the s_0 highest indices of weight l
  $j \leftarrow max\{0 \leq i < n, i \notin \mathbf{F}, \text{weight}(i) = l\}$
  $\mathbf{F} \leftarrow \mathbf{F} \bigcup \{j\}$
end for
return F

Algorithm 4 functions

function FreezeWeight$(n, l, e, f, \mathbf{r}, \mathbf{F})$
  $\mathbf{F}' \leftarrow FreezeWeights(n, l, f, \mathbf{r}, \mathbf{F})$
  $t_i \leftarrow \{i \in \mathbf{F}' | \text{weight}(i) = l\}$
  $t \leftarrow t_i + e$
  if $t < 0$ or $t > \binom{log_2(n)}{l}$ **then return** Failure
  end if
  for t times **do**
    $j \leftarrow max\{0 \leq i < n, i \notin \mathbf{F}, \text{weight}(r_i) = l\}$
    $\mathbf{F} \leftarrow \mathbf{F} \bigcup \{r_i\}$
  end for
return F
end function

function FreezeWeights$(n, l, f, \mathbf{r}, \mathbf{F})$
  for $f - |\mathbf{F}|$ times **do**
    $j \leftarrow max\{0 \leq i < n, i \notin \mathbf{F}, \text{weight}(r_i) >= l\}$
    $\mathbf{F} \leftarrow \mathbf{F} \bigcup \{r_j\}$
  end for
return F
end function

Following the triplet-tuned frozen set construction algorithm [24], we generate a total of 10200 new frozen sequences, given the 5G reliability sequence. The specified ranges for the triplet parameters are as follows: $s_0 \in [\![0, 16]\!]$, $s_1 \in [\![-15, 4]\!]$ and $s_2 \in [\![-15, 14]\!]$.

3.2.2 Network training and testing

We adopt the same MLP model as described in section 3.1. The hyperparameters were set empirically. The depth and the width of the network are respectively $l = 4$ and $h = 640$. The shortcut gap G as defined below is set to 2. The input of the dataset consists not only in the varying positions all along the dataset but in the whole sequence. Consequently, the input dimension is set to 1024. This has demonstrated better learning performances. Figure 3.6 illustrates the training and validation losses throughout epochs. It seems that the network is able to generalize to none seen sequences, although the training loss is not as low as the initial database loss.

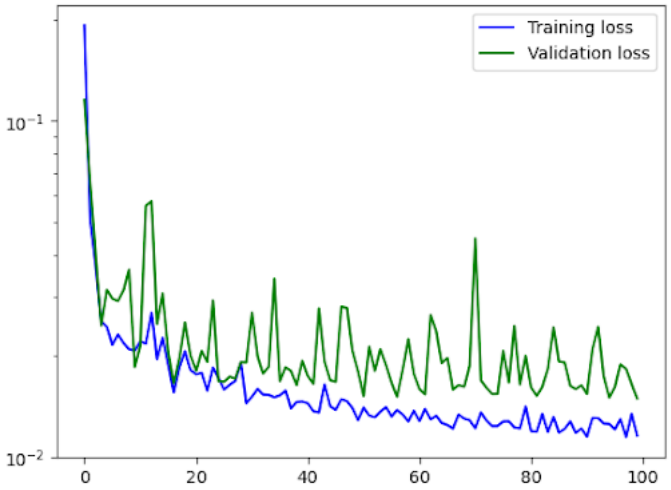

Figure 3.6: Training and validation losses throughout epochs

The test loss computed for the none seen sequences (10% of the dataset) amounts to 0.0157, which is lower than the loss observed using the initial dataset. The IOE metric is slightly worse compared to the first results. The structure of the new dataset is entirely different from the original one, necessitating training on the entire sequence, which increases the margin of error. Refer to Table 3.2 for a compilation of the testing results.

Test loss	Average IOE	Worst IOE
0.0157	0.1066	0.4091

Table 3.2: Test results

3.2.3 PGD method and results

The PGD method applied to the new trained network has generated some competitive sequences. The best sequence was generated within a number of iterations $I = 1500$ with a learning rate $\mu = 0.05$. The FER performance is illustrated in Figure 3.7. Although the new construction performance is better than the encountered sequences in the high SNR region, it does not outperform them by a significant margin. In fact, while the network has succeeded in producing new remarkable constructions, the constraints within the learning space have not facilitated notable improvements in performance.

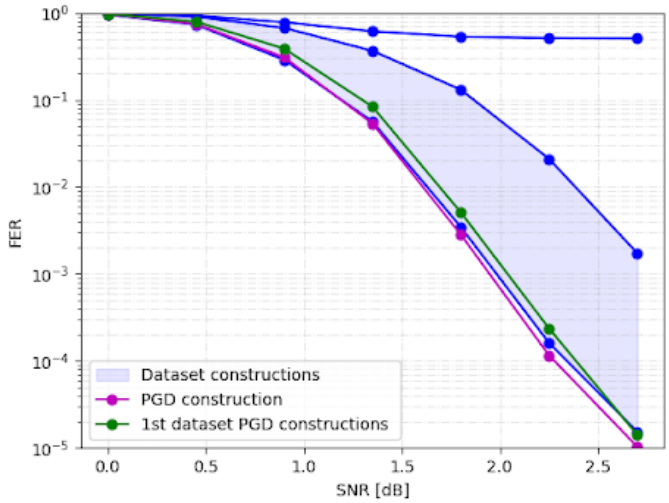

Figure 3.7: FER performance of PGD construction

3.3 Near-ML PAC Codes: Efficient Design with Controlled SCL Decoding Complexity

The previous sub-sections examined how learning techniques can be used to design polar codes with strong performance under a prescribed successive cancellation list (SCL) decoding algorithm. In this section, we shift our focus to designing improved polar-augmented convolutional (PAC) codes that can approach or even reach MLD under the same constraint of low-complexity SCL decoding, but adopting an expert-driven approach this time. PAC codes can be considered as a general class of precoded polar codes, obtained by applying a constant pre-transformation to all bits, both information and frozen. The problem of optimal code design for PAC codes remains an open research question that has been explored in numerous recent works [26–31]. Several of these studies [26, 27, 30], have included the weight distribution, and particularly the minimum weight codewords, into their design methodology due to its essential role in determining error-correction performance. This sub-section introduces a novel approach to design PAC codes that are guaranteed to achieve ML performance with moderate SCL list sizes. The method addresses the trade-off between error-correction performance and decoding complexity by jointly optimizing code distance properties and list size constraints. Unlike the heuristic and genetic-algorithm based methods that lack theoretical guarantees, the proposed approach is based on the binary decoding tree analysis along with the Difference to True Path Metric (DTPM) of possible paths. This enables to select frozen patterns that ensure low-complexity decoding while maintaining good distance properties.

The work described in this section has been submitted for publication at the IEEE international conference WCNC 2026 and is still under review at the time of this writing.

3.3.1 Polarization-Adjusted Convolutional (PAC)

Polarization-Adjusted Convolutional (PAC) codes [32] are a variant of polar codes that include a convolutional pre-transformation applied to the input vector $\mathbf{u}$ prior to encoding. This pre-transformation, defined by a convolutional generator polynomial $\mathbf{g} = (g_0, g_1, \ldots, g_{l-1})$, enhances the distance properties of the resulting code [33]. The generator matrix for PAC codes is given by :

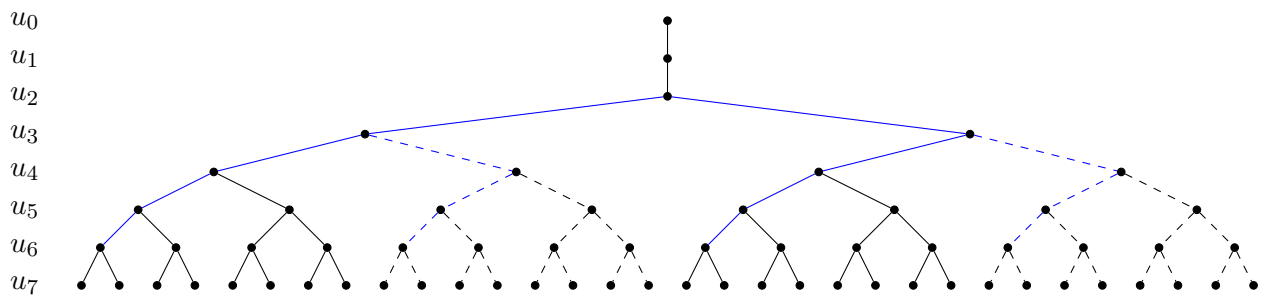

Figure 3.8: Binary tree for a (8,4) Reed-Muller (RM) constructed polar code, depicting all valid paths during SCL decoding. The left branches represent $u_i = 0$ and the right branches are $u_i = 1$. Pruned paths (dashed branches) correspond to invalid prefixes excluded by frozen-bit constraints. The blue paths are prefixes generating cosets in $\mathcal{U}_{\mathbf{f}_3^6} = \mathcal{U}_{0011}$ where $f_k = 1$ if position k is frozen; otherwise, $f_k = 0$.

$$\mathbf{G}_{\text{PAC}} = \begin{bmatrix} g_0 & g_1 & \cdots & g_{l-1} & 0 & \cdots & 0 \\ 0 & g_0 & \ddots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & g_1 & & \ddots & 0 \\ \vdots & & \ddots & g_0 & \ddots & & g_{l-1} \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & 0 & g_0 & g_1 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & g_0 \end{bmatrix} \mathbf{G}_n. \tag{3.6}$$

It should be noted that a conventional polar code is a special case of a PAC code where $\mathbf{g} = (1)$.

3.3.2 Successive Cancellation (SC) decoding

Under SC decoding, polar codes are provably capacity-achieving for asymptotically large block lengths N [34]. The SC decoder recursively computes log-likelihood ratios ($\mathcal{LLR}s$) at each stage i ($0 \leq i < N$) using the received vector $\mathbf{y}_0^{N-1}$ and previously decoded bits $\hat{\mathbf{u}}_0^{i-1}$. However, for finite block lengths, SC decoding is susceptible to error propagation because of its sequential nature. To overcome this limitation, successive cancellation list (SCL) decoding [35] maintains up to L candidate paths $\mathbf{u}_0^i$ at each decoding step i, where each path represents a valid prefix of the input sequence. By preserving multiple candidates, SCL decoding significantly improves error-correction performance compared to standard SC decoding. The decoding process can be viewed as a binary tree traversal of valid paths. An example of such a tree for a $(8, 4)$ code, constructed using the Reed–Muller (RM) code design [20], is given in Figure 3.8.

3.3.3 Difference to True Path Metric

SCL decoding approaches ML performance when the list size L is sufficiently large to ensure that the correct codeword remains among the list of candidates throughout the decoding process. The true path is dropped only when its path metric becomes worse than those of at least L other candidate paths. The Difference to True Path Metric (DTPM) introduced in [36] enables to evaluate this case. For a candidate path $\mathbf{u}_0^i$, and under the assumption that the all-zero codeword $\mathbf{0}_0^{N-1}$ is transmitted, the DTPM is defined as:

$$\psi_i(\mathbf{u}_0^i) = m_i(\mathbf{u}_0^i) - m_i(\mathbf{0}_0^i), \tag{3.7}$$

where $\mathbf{0}_0^i$ denotes the all-zero prefix and $m_i(\mathbf{u}_0^i)$ is the path metric for the candidate path $\mathbf{u}_0^i$. This metric simplifies to an Euclidean distance analysis by considering the coset characterization of PAC codes. A coset $\mathcal{C}_N(\mathbf{u}_0^i)$ represents the set of all codewords generated by the prefix $\mathbf{u}_0^i$:

$$\mathcal{C}_N(\mathbf{u}_0^i) = \left\{ \mathbf{c} = [\mathbf{u}_0^i, \mathbf{u}_{i+1}^{N-1}]\mathbf{G}_{PAC} \mid u_{i+1}^{N-1} \in \mathbb{F}_2^{N-1-i} \right\}. \tag{3.8}$$

Assuming that the closest codeword to $\mathbf{y}_0^{N-1}$ within a coset $\mathcal{C}_N(\mathbf{u}_0^i)$ is a minimum-weight codeword, the DTPM from (3.7) can be expressed as:

$$\psi_i(\mathbf{u}_0^i) = \min_{c \in \mathcal{C}^*} \frac{2}{\sigma^2} \sum_{j=0}^{N-1} y_j c_j, \tag{3.9}$$

where σ^2 is the noise variance and

$$\mathcal{C}^* = \left\{ \mathbf{c} \in \mathcal{C}_N(\mathbf{u}_0^i) \mid w^* = w(c) = w^*(\mathcal{C}_N(\mathbf{u}_0^i)) \right\}, \tag{3.10}$$

with $w^*(\mathcal{C}_N(\mathbf{u}_0^i))$ denotes the Hamming minimum-weight of the coset. An efficient algorithm for computing minimum weights of polar code cosets was introduced in [37] and was later generalized to PAC codes in [38].

Accordingly, the DTPM is the minimum of A^* correlated gaussian random variables:

$$\psi_i(\mathbf{u}_0^i) \sim \min \{X_0, \ldots, X_{A^*-1}\}, \tag{3.11}$$

where $X_k \sim \mathcal{N}\left(\frac{2w^*}{\sigma^2}, \frac{4w^*}{\sigma^2}\right)$ and A^* is the number of minimum-weight codewords within $\mathcal{C}^*$.

To ensure the correct codeword prefix at stage i remains in the list, all paths $\mathbf{u}_0^i$ with a negative DTPM must be retained. Consequently, the average list size L_i at stage i has to satisfy:

$$L_i \geq \sum_{\mathbf{u}_0^i \in \mathcal{U}_i} \mathbb{E}\left(\mathbb{1}\left(\psi_i(\mathbf{u}_0^i) \leq 0\right)\right) = \sum_{\mathbf{u}_0^i \in \mathcal{U}_i} \mathbb{P}\left(\psi_i(\mathbf{u}_0^i) \leq 0\right), \tag{3.12}$$

where $\mathcal{U}_i$ is the set of all valid prefixes at stage i and $\mathbb{1}$ is the indicator function. Empirically, it has been demonstrated that for moderate to long codes, restricting computations to cosets with minimum weights not exceeding — or are slightly greater than — the code's minimum distance d_{min} provides reliable estimates of the average list size.

In summary, the list size ($L = \max_i L_i$) required to achieve ML performance is fully characterized by the *Cumulative Distribution Function* (CDF) of the DTPM of candidate prefixes $\mathbf{u}_0^i$ at each decoding stage. This CDF depends solely in the minimum-weight properties of the coset $\mathcal{C}_N(\mathbf{u}_0^i)$ and the correlation factor ρ among its minimum-weight codewords. These properties could be efficiently computed thanks to the algorithms presented in [37] and [38] as explained in [36]. Building upon this work, we propose a code construction algorithm for PAC codes that simultaneously guarantees near-ML performance under low-complexity SCL decoding and good error-correction performance.

3.3.4 Polar code construction under SCL List size constraint

This section details the proposed algorithm for designing PAC codes that achieve ML performance under low-complexity SCL decoding. As the main objective of this method is to handle the trade-off between error-correction performance and decoding complexity, the algorithm selects codes that simultaneously satisfy two key constraints. First, a target minimum distance d_{target} to ensure strong error-correction capability. Second, a target list size L_{target} to guarantee near-ML performance with moderate decoding complexity.

Description of the proposed algorithm

Given code parameters such as the block length N, the code rate $R = \frac{K}{N}$ and the convolutional generator polynomial $\mathbf{g}$, the algorithm outputs frozen sequences that meet the specified constraints L_{target} and d_{target} .

The construction begins by freezing all bit positions corresponding to rows in the PAC generator matrix G_{PAC} with an Hamming weight less than d_{target}. This initial step immediately eliminates a subset of frozen sequences that fail to meet the minimum distance constraint. As a result, the search space is significantly reduced.

The algorithm fundamentally relies on the binary decoding tree traversal, starting from the first unfrozen bit position j. For subsequent stages $i \geq j$, the algorithm enumerates all valid candidate paths $\mathbf{u}_0^i$ corresponding to up to 2^{i-j+1} possible frozen sequences $f_j f_{j+1} \dots f_i \in \{0,1\}^{i-j+1}$. In these sequences, $f_k = 1$ indicates frozen bit position and $f_k = 0$ indicates information bit position. For each candidate frozen sequence $\mathbf{f}_j^i = f_j f_{j+1} \dots f_i$ at the i^{th} stage, the average list size $L_i^{\mathbf{f}_j^i}$ is computed using Equation (3.13).

$$L_i^{\mathbf{f}_j^i} = \sum_{\mathbf{u}_0^i \in \mathcal{U}_{\mathbf{f}_j^i}} \mathbb{P}\left(\psi(\mathbf{u}_0^i) \leq 0\right), \tag{3.13}$$

where $\mathcal{U}_{\mathbf{f}_j^i} = \left\{\mathbf{u}_0^i \mid w^*(\mathcal{C}_N(\mathbf{u}_0^i)) \leq d_{target}, u_k \in \mathbb{F}_{2-f_k}\right\}$, with $\mathbb{F}_1 = \{0\}$ and $\mathbb{F}_2 = \{0,1\}$.

To illustrate this principle, assuming no coset pruning is applied, paths in $\mathcal{U}_{f_3^6} = \mathcal{U}_{0011}$ are given in Figure 3.8. It should be noted that pruning strategies are detailed in Sub-section 3.3.4.

To ensure the minimum distance constraint, we leverage the properties of coset minimum-weight computation [37, 38]. At the last frozen stage, the minimum weight of a coset is lower bounded by the code's minimum distance, since the minimum weight among the remaining cosets at this stage determines the overall minimum distance of the code. Therefore, once the traversal reaches stages $i \geq N - K$, the minimum distance $d_{min}^{\mathbf{f}_j^i}$ of frozen patterns $\mathbf{f}_j^i$ with exactly $N - K$ frozen positions is computed from the set of valid paths $\mathcal{U}_{\mathbf{f}_j^i}$. If $d_{min}^{\mathbf{f}_j^i} \geq d_{target}$, the sequence is returned; otherwise it is discarded.

Pruning strategies

The exhaustive evaluation of all possible frozen patterns becomes computationally high in later stages as the number of frozen sequences grows exponentially. In order to keep the computational complexity reasonable, the algorithm employs pruning strategies at two levels.

A candidate frozen sequence is immediately discarded if it violates any of the following constraints:

- Rate constraint: the number of information bits or frozen bits is inconsistent with the code rate: $\sum_{k=j}^{i} f_k > N - K - j$ or $\sum_{k=j}^{i}(1 - f_k) > K$
- List size constraint: the average list size at stage i exceeds the target: $L_i^{\mathbf{f}_j^i} > L_{target}$

Additional constraints on the frozen patterns structure can be applied to remove potentially irrelevant ones. For instance, a constraint may require that, within a specific range of bit positions, a minimum number of information bits must be inserted by a certain decoding stage. Moreover, the last frozen bit position l_f may be predefined. This implies that all subsequent bits must be information bits. If the number of sequences exceeds a maximum value S_{max}, only the first S_{max} sequences are kept.

Pruning is applied at the level of path prefixes. A path $\mathbf{u}_0^i$ is pruned if the minimum Hamming weight of its associated coset $\mathcal{C}_N(\mathbf{u}_0^i)$ exceeds d_{target}. Those cosets are irrelevant for both the list size computation in equation (3.13) and the overall code minimum distance computation. Moreover, only paths in the sets $\mathcal{U}_{\mathbf{f}_j^i}$ of the retained sequences $\mathbf{f}_j^i$ are considered. In the last stages, the average list size is generally lower than the global list size needed to achieve ML performance. Consequently, it is possible to only consider cosets whose weights are less than the target minimum distance d_{target} in those stages. This simplification is valid because the algorithm's objective is to evaluate the code's minimum distance. Actually, the number of minimum-weight codewords is not taken into account.

The complete proposed algorithm is formalized in Algorithm 3.3.4.

Algorithm 1: Design of near-ML performance PAC codes with moderate SCL list size

Input: N, K, L_{target}, d_{target}, S_{max}, l_f, σ^2.
Output: frozen sequence meeting d_{target} and L_{target} constraints.

Pre-freeze all bit channels k where the row weight in G_{PAC} is $< d_{\text{target}}$;
$j \leftarrow$ first unfrozen bit index;
Initialize an empty stack $\mathcal{P}$ and push the prefixes 0_0^j and $(0_0^{j-1}, 1)$;
Initialize an empty list $\mathcal{F}$ and push the frozen sequences $f_j = 0$ and $f_j = 1$;
while $\mathcal{P}$ *is not empty* **do**
    Pop all prefixes $\mathbf{u}_0^i$ from $\mathcal{P}$;
    Compute w^*, A^* and ρ of each $\mathbf{u}_0^i$;
    if $i < l_f$ **then**
        $\mathcal{L} \leftarrow$ Next prefixes whose $w^* \leq d_{target}$;
    Compute and store $\mathbb{P}\left(\psi_i(\mathbf{u}_0^i) \leq 0\right)$ of each $\mathbf{u}_0^i$ using w^*, A^*, ρ and σ^2 ;
    $\mathcal{F}' \leftarrow \emptyset$;
    foreach *sequence* $\mathbf{f}_j^i \in \mathcal{F}$ **do**
        $L_i^{\mathbf{f}_j^i}, D_i^{\mathbf{f}_j^i} \leftarrow$ ComputeListSizeAndDmin($\mathbf{f}_j^i$);
        if $L_i^{\mathbf{f}_j^i} \leq L_{target}$ **then**
            if $\mathbf{f}_j^i$ *has exactly* $N - K$ *frozen positions and* $D_i^{\mathbf{f}_j^i} \geq d_{target}$ **then**
                return $\mathbf{f}_j^i$
            else
                foreach *posssible extension* $\mathbf{f}_j^{i+1}$ **do**
                    if $\mathbf{f}_j^{i+1}$ *meets rate and structural constraints* **then**
                        Push $\mathbf{f}_j^{i+1}$ into $\mathcal{F}'$;
    $\mathcal{F} \leftarrow \mathcal{F}'[1 : \min(|\mathcal{F}'|, S_{\max})]$;
    foreach $\mathbf{u}_0^{i+1} \in \mathcal{L}$ **do**
        Push $\mathbf{u}_0^{i+1}$ into $\mathcal{P}$ if it is a valid prefix of at least one sequence in $\mathcal{F}$;
return $\emptyset$ `/* If no valid sequence is found (or return codes with lower minimum distance) */`

Table 3.3: The distance properties and average SCL list sizes of the retained codes

K	Frozen Sequence	g	$d_{\min}$	$A_{d_{\min}}$	L
24	$Code_{24,1}$	$\mathbf{g}_1$	32	1516	16
		$\mathbf{g}_2$	32	264	16
	$Code_{24,2}$	$\mathbf{g}_1$	32	1836	11
		$\mathbf{g}_2$	32	396	10
	$5G - WD_{24}$	$\mathbf{g}_2$	32	176	110
	5G	$\mathbf{g}_2$	16	8	8
64	$Code_{64,1}$	$\mathbf{g}_2$	12	596	19
	$Code_{64,2}$	$\mathbf{g}_2$	12	608	28
	$5G - WD_{64}$	$\mathbf{g}_2$	16	2529 [27]	142
	5G	$\mathbf{g}_2$	8	256	23

Algorithm 2: ComputeListSizeAndDmin($\mathbf{f}_j^i$)

Input: Partial frozen sequence $\mathbf{f}_j^i$.

Output: Estimated list size $L_i^{\mathbf{f}_j^i}$.

$L_i^{\mathbf{f}_j^i} \leftarrow 0$;

Construct the set of valid prefixes $\mathcal{U}_{\mathbf{f}_j^i}$.;

foreach *path* $\mathbf{u}_0^i \in \mathcal{U}_{\mathbf{f}_j^i}$ **do**

  $L_i^{\mathbf{f}_j^i} \leftarrow L_i^{\mathbf{f}_j^i} + \mathbb{P}(\psi(\mathbf{u}_0^i) \leq 0)$;

$D_i^{\mathbf{f}_j^i} \leftarrow$ minimum weight among cosets in $\mathcal{U}_{\mathbf{f}_j^i}$;

return $L_i^{\mathbf{f}_j^i}, D_i^{\mathbf{f}_j^i}$

3.3.5 Simulation results

This section provides the simulation results for PAC codes with parameters $(128, 24)$ and $(128, 64)$, using convolutional transformations $\mathbf{g}_1 = (1, 0, 1)$ and $\mathbf{g}_2 = (1, 0, 1, 1, 0, 1, 1)$. A target FER of 10^{-3} is the common point for comparing the resulting code designs .

For the $(128, 24)$ code, Algorithm 3.3.4 was employed to identify many frozen sequences satisfying the constraints of a target minimum distance $d_{target} = 32$ and a maximum list size $L_{target} = 16$. Two promising codes, designated as $Code_{24,1}$ and $Code_{24,2}$, were selected for further comparison. The average list sizes Li required at each decoding stage i for both codes and convolutional transformations are plotted in Figure 3.9. For $Code_{24,1}$, a list size of $L = 16$ is sufficient to achieve its ML performance, with both $\mathbf{g_1}$ and $\mathbf{g_2}$. On the other hand, $Code_{24,2}$ demonstrates lower decoding complexity, requiring only $L = 11$ with $\mathbf{g_1}$ and $L = 10$ with $\mathbf{g_2}$ to reach its ML performance. These results are summarized in Table 3.3 along with the distance properties of both codes. The performance of $Code_{24,1}$ under SCL decoding is given in Figure 3.10. As expected from the list size analysis, the FER curves for $L = 16$ and $L = 32$ are almost identical, confirming that $L = 16$ is sufficient to achieve ML performance. Although the $5G\text{-}WD_{24}$ code [27] has better distance properties and therefore, potentially enhanced error-correction performance, it requires a much larger average list size of about $L = 110$ to achieve its ML performance. This is significantly higher than the optimal list size of both $Code_{24,1}$ and $Code_{24,2}$. On the other hand, the performance of $5G\text{-}WD_{24}$ with lower list sizes is worse than that of $Code_{24,1}$. Actually, its performance with $L = 32$ is matched by $Code_{24,1}$ with only $L = 8$, and its performance with $L = 64$ is surpassed by $Code_{24,1}$ with $L = 16$ using the same convolutional transformation $\mathbf{g}_2$. The 5G code performance is given for purposes of comparison. It demonstrates significantly worse ML performance than the proposed codes, which is attributed to its poor distance properties. While The 5G code achieves its ML performance with a small list size ($L = 8$), it is still outperformed by all other designs with this same decoding complexity. A direct comparison between $Code_{24,1}$ and $Code_{24,2}$ is illustrated in Figure 3.11. With a list size of $L = 8$, $Code_{24,2}$ not only outperforms $Code_{24,1}$ decoded with the same list size, but also nearly matches its performance with $L = 16$. This observation applies to both convolutional transformations $\mathbf{g_1}$ and $\mathbf{g_2}$.

For a code rate $R = \frac{1}{2}$, the algorithm parameters were set to $d_{\text{target}} = 16$ and $L_{\text{target}} = 32$. Only the transformation $\mathbf{g}_2$ was studied, since the PAC matrix of $\mathbf{g}_1$ suggests that only a RM code design provides a code minimum distance of 16. At this list size threshold, no codes with $d_{min} = 16$ were found. But, codes with $d_{min} = 12$ were retained. Two resulting codes, referred to as $Code_{64,1}$ and $Code_{64,2}$ are compared to the RM code design which matches the code $5G - WD_{64}$ of [27]. The average list sizes of both codes over the decoding stages are given in Figure 3.12. $Code_{64,1}$ needs a SCL list size of $L = 19$ to approach its ML performance while $Code_{64,2}$ demonstrates a slightly higher decoding complexity. A comparison of $Code_{64,1}$ and $Code_{64,2}$ with different SCL list sizes is presented in Figure 3.13. A list size of $L = 16$ for $Code_{64,1}$ achieves the same error rate performance as $Code_{64,2}$ with $L = 28$. $5G\text{-}WD_{64}$ code has better distance properties and potentially better error-correction performance if a large list size of about $L = 144$ is considered. However, its performance falls below $Code_{64,1}$ and $Code_{64,2}$ at lower list sizes as shown in Figure 3.14. $Code_{64,1}$ surpasses the performance

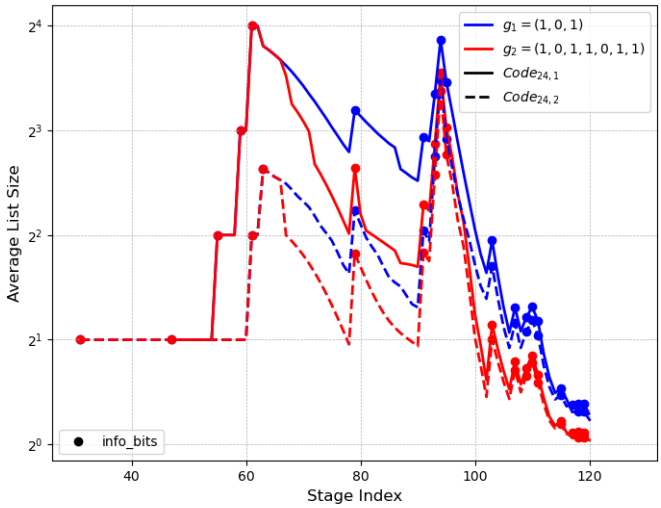

Figure 3.9: The average list sizes throughout the decoding stages for the proposed codes $Code_{24,1}$ and $Code_{24,2}$

of 5G-WD_{64} with $L = 8$ and matches its performance with $L = 32$ while requiring only a list size of $L = 16$. The performance of the 5G code, decoded with a list size of $L = 32$, is included for reference. As expected from its distance properties, its performance is comparable to that of the other codes achieved with a much smaller list size.

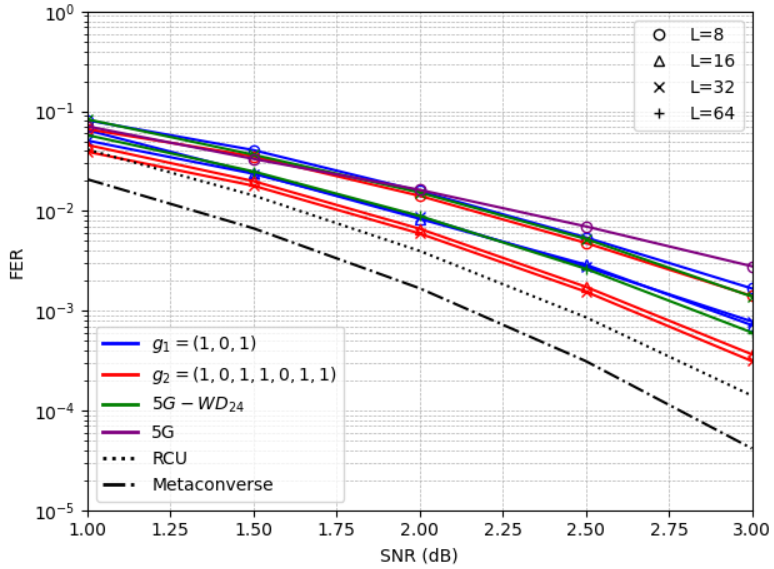

Figure 3.10: Error rate performance of $Code_{24,1}$

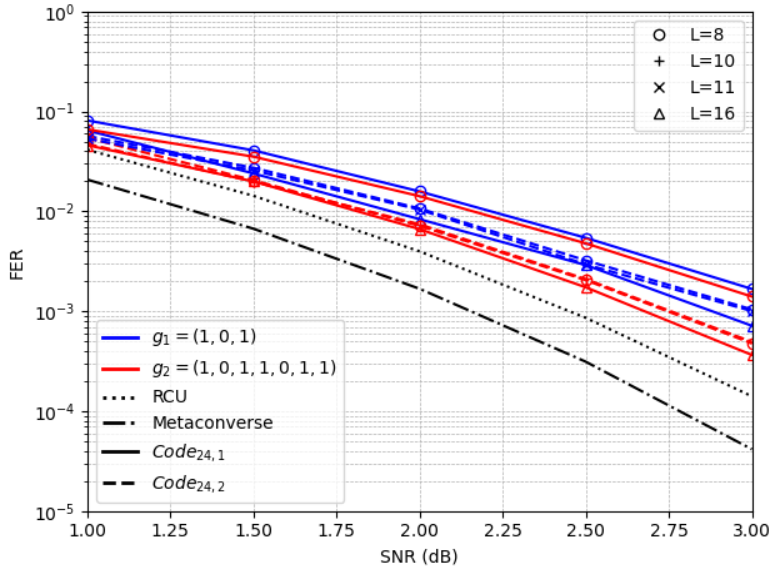

Figure 3.11: Error rate performance comparison between $Code_{24,1}$ and $Code_{24,2}$

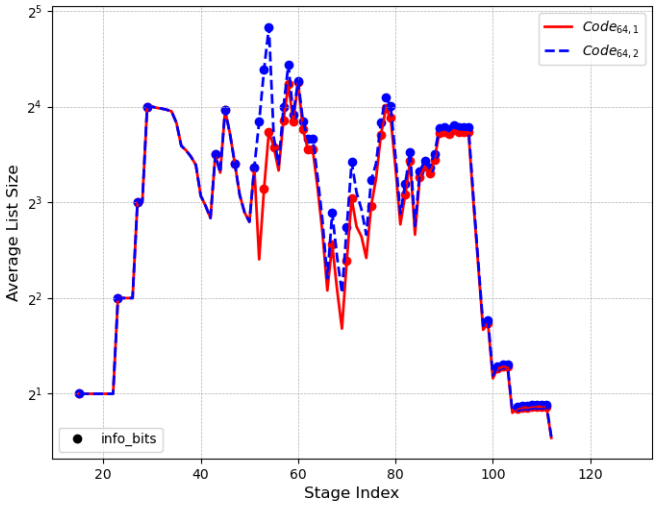

Figure 3.12: The average list sizes throughout the decoding stages for the proposed codes $Code_{64,1}$ and $Code_{64,2}$ with the convolutional transformation $\mathbf{g}_2$

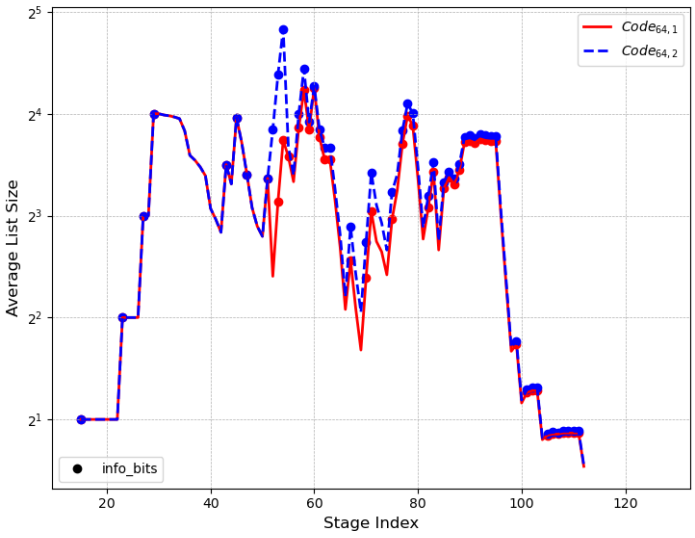

Figure 3.13: Error rate performance comparison between $Code_{64,1}$ and $Code_{64,2}$ with the convolutional transformation $\mathbf{g}_2$

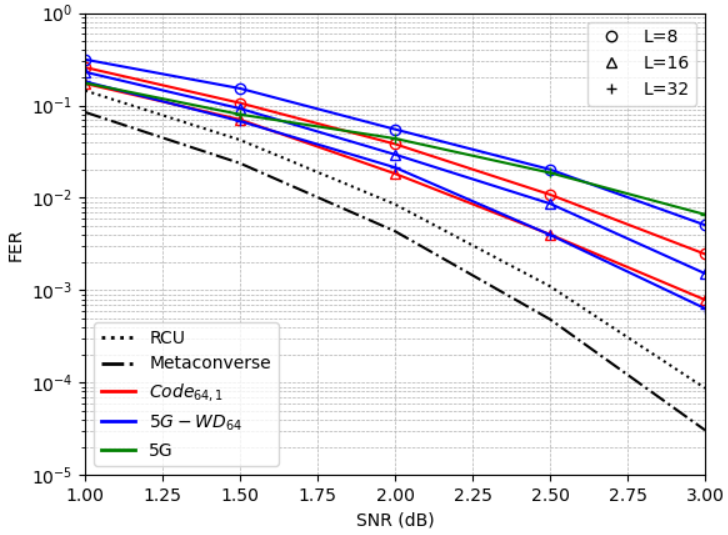

Figure 3.14: Error rate performance of $Code_{64,1}$

3.3.6 Summary

This section has introduced a new method for designing PAC codes that effectively balances the trade-off between decoding complexity and error-correction performance. The proposed approach, which optimizes the frozen sequence based on the minimum distance and list size constraints, successfully identifies configurations that achieve near-ML performance with significantly reduced SCL list size and consequently moderate decoding computational complexity. Results demonstrate that the generated codes approach the performance of other configurations with a list size half as large. This can be very useful for low-latency applications, where a balance between decoding complexity and error-correction capability has to be considered during code construction.

Conclusion

In conclusion, the general objective of this research was to improve polar code constructions specifically tailored for a (low-complexity) SCL decoder. Our first approach built upon the work done in [25], which involved training first a neural network to predict the Frame Error Rate (FER) for a given polar code construction, e.g. polar codes constructed from a gaussian approximation or RM polar sub-codes. Once trained, this network is used to generate better constructions by means of the PGD method. The resulting constructions were found to outperform the constructions of both datasets - both the original dataset utilized in [25] and the RM subcode dataset [24] - in the high SNR region. Although the margin of improvement remains modest, our research contributes to the ongoing efforts in advancing polar code constructions for SCL decoders by shedding light on the potential of combining neural network predictions with iterative refinement techniques like the PGD. It also highlights the importance of choosing well the database used to train the NN in charge of performance prediction. Our second contribution targeted the more powerful PAC codes. We have developed a novel method for designing polar codes under SCL decoding, ensuring near-ML error-correction performance with moderate list sizes (half-the-size of the usual standard decoder). The approach explicitly addresses the trade-off between error-correction performance and decoding complexity by jointly optimizing code distance properties and list size constraints. A promising avenue for future work is to investigate whether integrating machine learning into this method could further reduce the required list size while maintaining constant error-correction performance.

4 Conclusions and Perspectives

This deliverable aimed to summarize the investigations carried out within WP2 during the project. The goal of WP2 was to investigate how learning techniques can help come up with new codes or new design methods in communication scenarios for which existing methods fall short of the theoretical forecasts.

Two contributions have focused on revisiting existing design paradigms to arrive at better codes. In the first one, multi-agent deep reinforcement learning has been applied to the problem of optimizing the mapping between code symbols from a powerful non-binary turbo-code and modulated symbols from a high-order constellation. The result is a learned non-uniform constellation and a learned non-linear transformation that each improve the error-floor performance by one or two decades, compared to previous coded modulation designs based on expert rules. The second contribution leverages on supervised learning in an original manner, in order to revisit conventional Polar code design and arrive at improved performance under successive-cancellation list (SC) decoding. The learned polar codes are able to match and even slightly outperform the best existing construction, which is quite promising. However the performance of this approach was found to be highly dependent on the code database used for training. We have also introduced a new (expert) method for designing Polarization-Adjusted Convolutional (PAC) codes that effectively balances the trade-off between complexity (list size) and error-correction performance under SCL decoding.

Another line of work in WP2 aimed at using ML to learn new families of codes or communication waveforms. The third contribution fits within this framework. Here ML was used to search for a particular form of spread-spectrum sequences with good correlation properties, especially when truncated. Analysis of the learned outcomes gradually unveiled mathematical structure in the solutions, which eventually led to a deterministic, mathematical construction of optimal sequences. We foresee promising applications for these C4-sequences that have been patented, particularly for massive Internet-of-Thing (IoT) and non-terrestrial network (NTN) communications systems.

Bibliography

[1] D. Chu, "Polyphase codes with good periodic correlation properties (corresp.)," *IEEE Transactions on Information Theory*, vol. 18, no. 4, pp. 531–532, 1972.

[2] R. Frank, S. Zadoff, and R. Heimiller, "Phase shift pulse codes with good periodic correlation properties (corresp.)," *IRE Transactions on Information Theory*, vol. 8, no. 6, pp. 381–382, 1962.

[3] C. Marchand and E. Boutillon, "Rate-adaptive inner code for non-binary decoders," in *2021 11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1–5.

[4] ——, "Rate-adaptive cyclic complex spreading sequence for non-binary decoders," in *International Symposium on Topics in Coding (ISTC'2023), Brest*, 2023.

[5] E. Boutillon, "C4-sequences: Rate adaptive coded modulation for few bits message," in *International Symposium on Topics in Coding (ISTC'2023), Brest*, 2023.

[6] AI$_4$CODE Project (ANR-21-CE25-0006), "Specification of the code design problems," Deliverable D2.1, Apr. 2023.

[7] R. Klaimi, C. Abdel Nour, C. Douillard, and J. Farah, "Design of Low-Complexity Convolutional Codes over GF(q)," in *IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, UAE, Dec 2018.

[8] R. Klaimi, S. Weithoffer, and C. Abdel Nour, "Improved non-uniform constellations for non-binary codes through deep reinforcement learning," in *IEEE 23rd Int. Workshop on Signal Process. Advances in Wireless Commun. (SPAWC)*, 2022, pp. 1–5.

[9] R. Klaimi, "Study of non-binary turbo codes for future communication and broadcasting systems," Theses, IMT Atlantique, Jul. 2019. [Online]. Available: https://theses.hal.science/tel-02543195

[10] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[11] H. Dong, H. Dong, Z. Ding, S. Zhang, and Chang, *Deep Reinforcement Learning*. Springer, 2020.

[12] R. Garzón-Bohórquez, C. Abdel Nour, and C. Douillard, "Protograph-based interleavers for punctured turbo codes," *IEEE Trans. Commun.*, vol. 66, no. 5, pp. 1833–1844, 2018.

[13] R. Garzón Bohórquez, R. Klaimi, C. Abdel Nour, and C. Douillard, "Mitigating correlation problems in turbo decoders," in *10th Intern. Symp. on Turbo Codes Iter. Inf. (ISTC)*, Hong Kong, China, Dec. 2018.

[14] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[15] Y. Liao, S. A. Hashemi, J. M. Cioffi, and A. Goldsmith, "Construction of polar codes with reinforcement learning," *IEEE Transactions on Communications*, vol. 70, no. 1, pp. 185–198, 2021.

[16] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, 2013.

[17] A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Genetic algorithm-based polar code construction for the awgn channel," in *SCC 2019; 12th International ITG Conference on Systems, Communications and Coding*. VDE, 2019, pp. 1–6.

[18] L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "Ai coding: Learning to construct error correction codes," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 26–39, 2019.

[19] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Trans. on Comm.*, vol. 60, no. 11, pp. 3221–3227, 2012.

[20] B. Li, H. Shen, and D. Tse, "A rm-polar codes," *arXiv preprint arXiv:1407.5483*, 2014.

[21] G. He, J.-C. Belfiore, I. Land, G. Yang, X. Liu, Y. Chen, R. Li, J. Wang, Y. Ge, R. Zhang *et al.*, "Beta-expansion: A theoretical framework for fast and recursive construction of polar codes," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[22] L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "Reinforcement learning for nested polar code construction," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[23] M. Léonardon and V. Gripon, "Using deep neural networks to predict and improve the performance of polar codes," in *2021 11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1–5.

[24] V. Miloslavskaya, Y. Li, and B. Vucetic, "Design of compactly specified polar codes with dynamic frozen bits based on reinforcement learning," 2022, submitted to IEEE Transactions on Communications.

[25] M. Léonardon and V. Gripon, "Using deep neural networks to predict and improve the performance of polar codes," in *2021 11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1–5.

[26] M. Rowshan, S. H. Dau, and E. Viterbo, "On the formation of min-weight codewords of polar/pac codes and its applications," *IEEE Transactions on Information Theory*, vol. 69, no. 12, pp. 7627–7649, 2023.

[27] M. Rowshan and V.-F. Dragoi, "Towards weight distribution-aware polar codes," *arXiv preprint arXiv:2506.15467*, 2025.

[28] M. Moradi and D. G. Mitchell, "Pac code rate-profile design using search-constrained optimization algorithms," in *2024 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2024, pp. 2204–2209.

[29] M. Moradi and A. Mozammel, "A monte-carlo based construction of polarization-adjusted convolutional (pac) codes," *Physical Communication*, vol. 68, p. 102578, 2025.

[30] X. Gu, M. Rowshan, and J. Yuan, "Pac codes meet crc-polar codes," *arXiv preprint arXiv:2501.18080*, 2025.

[31] W. Liu, L. Chen, and X. Liu, "A weighted sum based construction of pac codes," *IEEE Communications Letters*, vol. 27, no. 1, pp. 28–31, 2022.

[32] E. Arıkan, "From sequential decoding to channel polarization and back again," *arXiv preprint arXiv:1908.09594*, 2019.

[33] B. Li, H. Zhang, and J. Gu, "On pre-transformed polar codes," *arXiv preprint arXiv:1912.06359*, 2019.

[34] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[35] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "Llr-based successive cancellation list decoding of polar codes," *IEEE transactions on signal processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[36] M. Ellouze, "Distance properties of polar codes: theory and applications," Ph.D. dissertation, Université de Bordeaux, 2024.

[37] H. Yao, A. Fazeli, and A. Vardy, "A deterministic algorithm for computing the weight distribution of polar code," *IEEE Transactions on Information Theory*, 2023.

[38] M. Ellouze, R. Tajan, C. Leroux, C. Jégo, and C. Poulliat, "Low-complexity algorithm for the minimum distance properties of pac codes," in *2023 12th International Symposium on Topics in Coding (ISTC)*. IEEE, 2023, pp. 1–5.