RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Doctorat de l'Université de Toulouse

délivré par l'ISAE-SUPAERO

Solutions de décodage canal basées sur l'apprentissage automatique pour les communications de type machine-à-machine

---

Thèse présentée et soutenue, le 13 décembre 2024 par

Gastón DE BONI ROVELLA

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
ISAE-ONERA SCANR Signal communication Antennes Navigation Radar

Thèse dirigée par
Jérôme LACAN et Meryem BENAMMAR

Composition du jury
M. Raphaël LE BIDAN, Rapporteur, IMT Atlantique Brest
M. Maël LE TREUST, Rapporteur, Université de Rennes, CNRS, INRIA/IRISA
Mme Inbar FIJALKOW, Examinatrice, ENSEA/CY Cergy Paris Université/CNRS
M. Charly POULLIAT, Examinateur, Toulouse INP - ENSEEIHT
M. Jérôme LACAN, Directeur de thèse, Institut Supérieur de l'Aéronautique et de l'Espace
Mme Meryem BENAMMAR, Co-directrice de thèse, Institut Supérieur de l'Aéronautique et de l'Espace

Membres invités
M. Hugo MERIC, Centre Nationale d'Études Spatiales
M. Tarik BENADDI, Thalès Alénia Space

Acknowledgments

The following hundred-and-something pages you are about to read are but the result of all the people in my life who have consistently shown support, affection, encouragement, comfort, and everything else that gives one the strength to carry on. As unfair and insufficient as it certainly is, I will do my best to express my gratitude in only a few paragraphs. I apologize in advance for anyone I may forget.

I would like to start by thanking the jury, M. Raphaël Le Bidan, M. Maël Le Treust, Mme Inbar Fijalkow and M. Charly Poulliat, who kindly agreed to take part in this process. Their comments, questions, and suggestions far exceeded my expectations, and their warm and generous words filled my inner motivation battery for at least a couple of years.

Let me continue with my Ph.D. supervision team. I thank my directors, Meryem Benammar and Jérôme Lacan, for the incredible opportunity and for all the help and guidance they have provided throughout the past three years. This work would not have been possible without them. I also thank my co-supervisors, Hugo Méric from and Tarik Benaddi, for their valuable remarks and suggestions during our meetings.

On a more personal level, four great pillars have managed to support me with the ever-increasing weight that a Ph.D. carries, and I believe each one deserves —at least— a special mention.

First, the ISAE-SUPAERO team. The ComIT research group welcomed me to their trenches, and the rest of the Ph.D. students from the DEOS department made sure I never lacked company for my morning coffee. *Les vieux* (Damien, Stéphanie, Wallance, José, and Meryem) who made the best efforts to keep a friendly and welcoming workplace, with regular team meals and the occasional —strictly professional— pool party. *Les jeunes* (Benjamin, Fabio, Khaled, Leila, Steven, Cyril, Florent, Antoine, César, Thomas, Javier, Agathe, Martin, Asnate, Joanna, Marco, Sarah, Titouan, Max, and others), that constitute the social backbone that every Ph.D. student needs to maintain sanity and good spirits. I must reserve at least one line to thank Nadia, who put up with my nonsense throughout my entire time in SUPAERO, and who has promised (very seriously) to come visit me in Uruguay. From now on, and as long as this document exists, she is welcome anytime.

Second, we have the TéSA family. Why did I get assigned to this lab, where there is not one single person associated with my Ph.D.? The question could open another field of research. But I am deeply thankful it turned out this way. Corinne, I think your office is too small for the size of your heart. Evelyne, Hamish and Youssra, thank you for doing such great theses, it is no pressure for me at all. Finally, thanks to all the rest of the people who make up this family-lab: Esteban, Raoul, Philippe, JY, Serge, Samy, Kareth, Joan, Marta, Valérian, Maurine, Jeff, Jihanne, Younes, Paul and Léa. Only one thing: next time, let *me* choose the movie.

Third, to my family, that group of stubborn, truthful, unbending, and trustworthy individuals that have carved me into who I am today. They have carried me in these past three years in more ways than they probably know. My fearless sister, I owe you the courage. My brilliant brother, I owe you the perspective. My kind father, I owe you the empathy. My loving mother, I owe you everything.

And finally, there is this girl that I have been dating for… ten years? I think it's getting serious. I suppose there is not much I could say to Flo in a few words that I haven't said since I was 16. I am sorry for being away for so many years. If you let me, I will spend the rest of our lives trying to make it up to you.

Contents

List of Figures

List of Tables

List of Acronyms

3GPP	*3rd Generation Partnership Project*
API	*Application Programming Interface*
AWGN	*Additive White Gaussian Noise*
BCH	*Bose–Chaudhuri–Hocquenghem*
BEP	*Bit Error Probability*
BER	*Bit Error Rate*
BICM	*Bit-Interleaved Coded Modulations*
BP	*Belief Propagation*
BPSK	*Binary Phase-Shift Keying*
BSC	*Binary Symmetric Channel*
CM	*Coded Modulations*
DNN	*Deep Neural Network*
ECCT	*Error Correction Code Transformer*
FEC	*Forward Error Correction*
FEP	*Frame Error Probability*
FER	*Frame Error Rate*
FFNN	*Feed-Forward Neural Network*
GF	*Galois Field*
GNN	*Graph Neural Networks*
GRAND	*Guessing Random Additive Noise Decoding*
GRU	*Gated Recurrent Unit*
iid	*independent and identically distributed*
IoT	*Internet of Things*
KKT	*Karush–Kuhn–Tucker*
LDPC	*Low-Density Parity-Check*

LLR	*Log-Likelihood Ratio*
LSTM	*Long Short-Term Memory*
M2M	*Machine-to-Machine*
MAP	*Maximum A Posteriori*
MH-SA	*Multi-Head Self-Attention*
MI	*Mutual Information*
ML	*Maximum Likelihood*
MLB	*Maximum Likelihood Bound*
MLP	*Multi-Layer Perceptron*
MPMI	*Mean Pairwise Mutual Information*
MRI	*Most Reliable Independent*
NN	*Neural Network*
OSD	*Ordered Statistics Decoder*
PHY	*Physical layer*
pdf	*probability density function*
PSK	*Phase-Shift Keying*
QAM	*Quadrature Amplitude Modulation*
QPSK	*Quadrature Phase-Shift Keying*
r-ECCT	*Recurrent Error Correction Code Transformer*
RBF	*Radial Basis Function*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent Neural Networks*
SBND	*Syndrome-Based Neural Decoder*
SC	*Successive Cancellation*
SCL	*Successive Cancellation List*
SGD	*Stochastic Gradient Descent*
SNR	*Signal-to-Noise Ratio*
SVM	*Support Vector Machine*

Notations

Random variables and vectors are represented using italic capital letters (e.g., X and $\boldsymbol{X}$), whereas Roman and bold letters (e.g., x and $\boldsymbol{x}$) denote their respective realizations. Matrices are represented by italic capital letters (e.g. A), and are always defined in the body of the text to prevent ambiguity. Additional notations are provided in the following list.

Symbol	**Description**
I_n	$n \times n$ identity matrix.
$\odot$	Hadamard product.
$\|\mathcal{X}\|$	Cardinality of the finite set $\mathcal{X}$.
$P_X(x)$	Probability distribution of X evaluated in x.
$P_{X,Y}(x,y)$	Joint probability distribution of (X,Y) evaluated in (x,y).
$P_{X\|Y}(x\|y)$	Conditional probability distribution of X given Y evaluated in (x,y).
$X \leftrightarrow Y \leftrightarrow Z$	Markov chain meaning $P_{Z\|Y,X} = P_{Z\|Y}$.
$\mathbb{P}(X = x)$	Probability of the event $\{X = x\}$.
$\mathbb{1}(e)$	Indicator of the event $\{e\}$ that takes the value 1 if and only if the event $\{e\}$ is true, and 0 otherwise.
$[1 : n]$	Set of integers from 1 to n.
$\lceil \cdot \rceil$	Ceiling function.
$\lfloor \cdot \rfloor$	Floor function.
$\oplus$	Exclusive OR operation.
$\mathcal{R}(x)$	Real part of the complex number x.
$\mathcal{I}(x)$	Imaginary part of the complex number x.

Introduction

Context

These last few decades have been witnesses to unprecedented advances in the field of wireless communications, including the fifth-generation (5G) mobile communication standard developed by the 3GPP [3GP18], which seeks to provide massive broadband seamless connectivity along with ultra-reliable and low latency, with relatively low complexity. This is empowered by enabling further development in communication protocols such as the Internet of Things (IoT), Machine-to-Machine (M2M) type communications, and more [Jia+17; ZPH19]. However, due to the particular latency and complexity constraints for such new protocols —M2M and IoT— the design of new-generation communication systems calls for more intricate and adaptive schemes than the classical solutions. The main limitations of the classical communication techniques are that (i) they are often *block-wise* designed, (ii) they are based on asymptotic optimality results, and (iii) they are very dependent on accurate modeling of the communication scenario.

As an alternative to these classical schemes, machine learning-based solutions are currently being explored extensively as enhancers or substitutes for many components of the communication chain [Eld+22; Sim18], focusing on applications in the physical layer (PHY) [Ye+24] (channel coding, modulation and waveform design, channel estimation and equalization, MIMO precoding and decoding, resource allocation, etc.) but also the access and network layers [Ahm+20; MLL19]. The main advantages of machine learning-based solutions are that they can allow for an *end-to-end* design, they may not require an accurate model of the communication scenario, and they can *adapt* to varying communication conditions. As such, these solutions are foreseen to play an even more prominent role in the upcoming 6G mobile communication standard, which will require even higher data rates and reliability, lower latency, and an increased flexibility compared to 5G [Row+24; She+20]. Besides, beyond the mobile communications settings (5G, 6G), recent works have also reported potential enhancements enabled by machine learning-based solutions in satellite communications, where their applications include interference detection, flexible payload configuration, and congestion prediction [Vaz+21]. Machine learning-based solutions are also being studied in optical communications settings as a way of dealing with non-linearities, software-defined networks, and parameter estimation, among other use cases [Kha+19; Zib+16].

As an essential —and often time-consuming— part of the communication system, channel coding for next-generation mobile communications (M2M and IoT) calls for the use of shorter block lengths, as they possess lower latencies compared to codes with longer block lengths. However, classical channel coding schemes carry two major drawbacks: (i) they are less reliable under non-asymptotic block lengths; and (ii) optimal decoders for short block lengths usually have a large computational complexity, leading to higher latency [Row+24]. The design of optimal and low-complexity decoders for short codes is thus of crucial importance for next-generation communication standards.

Early channel coding solutions using machine learning (more specifically, neural networks) were presented over thirty years ago [YBW89]. In 1989, a neural network was proposed as a Maximum Likelihood (ML) decoder for generic block codes [ZHA89]. In the same year, Bruck and Blaum determined the equivalence between finding the global maximum of a neural network function and ML decoding [BB89], even though the problems were recognized as NP-hard, and hence, intractable for the code lengths employed in realistic communication scenarios. These first works constituted only a proof-of-concept for deep learning-based channel coding solutions due to the lack of maturity of deep learning solutions and the limitation in the available computational power. However, the recent advances in deep learning triggered a renewal of interest in machine learning-based channel coding with the work of Gruber et al. [Gru+17] in 2017 which built a quasi-optimal neural network decoder for a very short Polar code. This constituted a turning point in the literature after which the design of machine learning-based channel coding schemes was massively undertaken by both the scientific and the industrial research communities [Lim+24; Niu+21].

Curse of dimensionality and contributions

Gruber et al. [Gru+17] trained a Multi-Layer Perceptron (MLP) to learn to decode very short binary linear block codes, namely a $(16, 8)$ Polar code [Ari09] and a $(16, 8)$ random code. This led to two very important conclusions. The first conclusion is that Deep Neural Networks (DNN) —such as an MLP— can fully learn to decode a linear code with close-to-optimal error probability when trained over the set of all valid codewords. However, this result remains valid only for very short codes, for which training over all valid codewords is practically possible. As the code length increases, the network's learning capacity rapidly shrinks. The second conclusion of [Gru+17] is that training a DNN on only a subset of the valid codewords (e.g., 70% of the codeword space) entails a performance loss for both random and structured codes. However, for random codes, the resulting error probability on the *unseen* codewords during training corresponds to that of a random guess decoder, whereas for the structured code (Polar code), the error probability of the unseen codewords outperforms that of a random guess. Hence, the more structured the code, the better the generalization capability of the DNN-based decoder.

These two conclusions establish the *curse of dimensionality* —originally introduced by Wang in 1996 while implementing a neural-based Viterbi decoder [WW96]— which constitutes a major challenge for the scientific communities that study machine learning for channel decoding. The curse of dimensionality (also referred to as *scalability problem*) is due to three main components. First, for an (n, k) channel code, the codeword space is composed of 2^k valid codewords of n bits. This exponential growth engenders a training dataset size exponentially increasing in k, which becomes intractably large to be fully explored even for short-to-medium length codes. Second, in order to prevent over-fitting to the noise realizations, the training dataset should allow to *see* each codeword with a variety of noise realizations, which expands further the size of the training dataset. Last but not least, the larger the block length of the code, the larger the structure of the neural network needed to properly learn the decoding function, inducing thus an increase in the number of trainable parameters, in the number of training epochs and, often, in the batch size. In the following, we detail our contributions in tackling each of these manifestations of the curse of dimensionality.

1) Number of noise realizations: bitwise SVM

One important aspect of the curse of dimensionality due to noise realizations pertains to the set of *correctable* noise realizations. To exemplify this, let us consider an (n, k) block linear code used to communicate over a Binary Symmetric Channel (BSC). As the code length n increases and the code rate $R = {}^{k}\!/_{n}$ decreases, an optimal decoder is able to correct a larger amount of bitflip patterns that affect the transmitted signal. For instance, for a code of length $n = 8$ and with an error correction capability of $t = 1$ bit, there are 8 correctable bitflip patterns[1]. In contrast, for a code of length $n = 100$ and an error correction capability of $t = 5$ bits, the number of correctable bitflip patterns is equal to:

$$\sum_{i=1}^{5} \binom{100}{i} \approx 79 \times 10^6 \tag{1}$$

This imposes an ever-growing number of possible noisy inputs that should be decoded as the same codeword, and the same is valid for every possible codeword.

To tackle the curse of dimensionality due to the number of noise realizations, we explore Support Vector Machines (SVM). SVMs are one of the most studied machine learning-based approaches in the literature [AML12; Sal+14], and were introduced over thirty years ago by Vapnik, Boser, Cortes, and Guyon [BGV92; CV95; Vap97]. They belong to the supervised learning methods, and are appealing for several reasons. First, they exhibit the so-called *maximum margin property*, since they entail a binary classifier which divides the dataset with a hyperplane that is at an equal and maximum distance from both classes. This will allow us to reduce the number of noise realizations for each class (codeword). Second, their training consists in a convex optimization problem, which ensures the convergence to a global minimum. Third, the mathematical construction of the classifier exploits usually a smaller subset of the dataset (called the *support vectors*) to produce the classifying function. For these reasons, SVMs have been widely employed in many communication applications, such as channel estimation [APP20; Gar+06], physical layer security [HDL19], multiuser detection [GK99], channel equalization [Gia+18; LHL05], and wireless signal identification [Tek+19], among others.

Previous studies have addressed the application of SVMs in channel decoding, as in [DH10; KB08; SY16]. For instance, [DH10] applies SVMs to adaptive modulation and coding in real-time scenarios, achieving significant complexity reductions over prior algorithms. In [KB08], the authors introduce a pairwise SVM-based decoder that demonstrates competitive perfor-

[1]There are probably a few more, but for simplicity, we only consider the patterns with at most t bitflips.

mance for convolutional codes, though at the cost of significant computational complexity. A similar approach is observed in the works of [Ram+09] and [SY16], where minor adjustments to training and application are made while adhering to the same pairwise classification strategy. In this method, however, both the necessary training dataset and the number of binary classifiers that constitute the decoder grow exponentially with the size of the code. This results in an intractable decoder when applied to a code length that is larger than a few bits. The largest implemented code has a length of $n = 15$ bits [SY16], and employs up to 100 noisy realizations of each valid codeword.

In this thesis, we introduce a novel bitwise SVM-based decoding framework that greatly reduces the number of binary classifiers to generate from 2^k to k for a channel code of parameters (n, k). Additionally, the maximum margin property is exploited in order to reduce the training dataset size to only one noiseless sample per valid codeword, as opposed to previous contributions which employed several noisy realizations of every possible codeword. We conclude by proving mathematically the equivalence between the proposed SVM decoder and the ML decoder in an Additive White Gaussian Noise (AWGN) channel. Hence, even if the complexity is greatly reduced with respect to previous SVM-based channel decoders, the curse of dimensionality pertaining to the size of the codeword space remains, which led us to the exploration of deep learning-based solutions.

2) Number of valid codewords: message-oriented model-free decoder

The SVM-based decoding framework, as we have approached it, is not able to learn properly without employing at least one sample of each class, i.e., of each possible codeword. This constitutes the second element of the curse of dimensionality, where the number of possible codewords increases exponentially with the code dimension (see Section 1.1). Hence, our motivation is to explore solutions that learn a decoding function without the need to explore the entire codeword space.

DNNs are well-known universal approximators [HSW89], which means that, when properly constructed, they are able to approximate any continuous function. Additionally, through optimization algorithms such as Stochastic Gradient Descent (SGD) [Ros58], this function can be obtained solely through input and output data of the desired function, i.e., the training dataset. This is why previous works have implemented this type of network as potential decoders but have faced the scalability problem [Gru+17; WW96]. Thus far, there are two main approaches present in the literature that undertake this problem and produce neural-based decoders with reasonable performances while only being trained on a subset of the valid codewords. These systems will be henceforth referred to as *scalable* neural decoders.

The first scalable neural decoder is a *model-based* approach, introduced by Nachmani et al. in 2016 [NBB16], which proposes a neural extension of the well-known Belief Propagation (BP) decoding algorithm [RL09]. It consists in, first, assigning weights to the edges of the Tanner graph on which the BP algorithm iterates through, and then training the resulting neural network to minimize the error rate using a stochastic gradient descent algorithm. Due to the iterative nature of the BP algorithm, [NBB16] resorts to a *deep unfolding* procedure, where the Tanner graph is unfolded into a DNN with twice as many layers as iterations employed, and the links between the layers abide by the structure of the code's bipartite graph. The main advantage of this technique is that it preserves the message passing symmetry conditions of the original BP, and as a consequence, the error rate is independent of the transmitted codeword. Therefore, it suffices to train the network with noisy observations of a single codeword, instead of the entire codeword space. This technique has been shown to partially reduce the negative effect related to the presence of short cycles in the Tanner graph, and systematically improve performances of the BP algorithm in dense codes. This work led to further advances in this approach that target lower complexity and better performances, including the implementation of Recurrent Neural Networks (RNN) [Nac+17; Nac+18], autoregressive architectures [NW21], advanced loss and multi-loss functions [LG18; Nac+18], Graph Neural Networks (GNN) [SW20], alternative labeling schemes [Che+23], and even the extension to other structures like factor graphs in Polar coding [Ari09; Xu+17].

Nachmani's model-based approach was proposed as an improved BP decoding for short-to-medium length BCH codes, which are high-density parity-check codes and thus prone to short cycles in their Tanner graph. Indeed, in that scenario, weighted BP produces better decoding performances than the classical BP algorithm. However, BP decoding is well-suited for very large and sparse codes —such as Gallager's Low-Density Parity-Check (LDPC) codes [Gal63]— due to the absence of short cycles in the bipartite graph induced by the parity-check matrix of the code. This minimizes the message correlation in the iterative algorithm, which in turn enables close-to-optimal decoding performances. In contrast, when applied to short or medium-length and dense codes —namely, BCH codes—, BP performs poorly compared to maximum likelihood decoding. Even if neural BP can partially alleviate the performance deterioration caused by the presence of short cycles in the Tanner graph, its impact remains moderate. Thus far, and to the best of our knowledge, no model-based implementations have been proposed that display close-to-optimal performances for the aforementioned types of codes.

The second technique consists of a *model-free* approach, introduced by Bennatan et al. in 2018 [BCK18a] and is tailored for transmissions over a Binary Input-AWGN (BI-AWGN) channel. In this scenario, the proposed decoder seeks to estimate the positions of the Binary Phase-Shift Keying (BPSK) symbols in the modulated codeword that have suffered a change in their sign, or analogously, in the binary domain, a *bitflip*. A preprocessing stage is added before the decoder, where the received signal $\boldsymbol{y}$ is divided into its absolute value $|\boldsymbol{y}|$ (also called *reliabilities*) and its syndrome $\boldsymbol{s}$, defined as:

$$\boldsymbol{s} = H\boldsymbol{y}^b = H(\boldsymbol{c} \oplus \boldsymbol{e}^b) = H\boldsymbol{e}^b, \tag{2}$$

where H denotes the parity-check matrix of the code, $\boldsymbol{y}^b$ represents the hard decision associated with the received signal $\boldsymbol{y}$, $\boldsymbol{c}$ indicates the transmitted binary codeword and $\boldsymbol{e}^b$ is a binary vector containing ones in the flipped positions and zeros everywhere else —also referred to as the bitflip pattern. Authors in [BCK18a] proved that these two elements $(H\boldsymbol{y}^b, |\boldsymbol{y}|)$ are sufficient statistics for the optimal estimation of the bitflip pattern $\boldsymbol{e}^b$ and are, by definition, independent of the transmitted codeword. This implies that (i) optimal decoding can be achieved with this framework; and (ii) training can be carried out using noisy observations of a single codeword, equivalently to the model-based approach. The original work by Bennatan et al. employed MLP and RNN as estimators for the bitflip positions, while further works have also used transformer-based architectures [CW22b; Par+23] and other iterative approaches [CW23; KP20]. Thus far, the contributions in the model-free approach have been focused on employing new NN-based architectures to obtain better results with lower complexities.

The model-free approach, though harder to interpret than the model-based, has shown competitive performances when applied to Polar and BCH codes, and approaching optimal error rates in some circumstances. However, the work in [BCK18a] presented some limitations that could be enhanced to further reduce the error rates: (i) the decoder estimates the entire codeword, assigning equal importance to every bit, regardless of whether it is an information bit or a parity bit; (ii) the application of the decoder to non-systematic codes implies further linear operations on the estimated codeword that leads to a penalty in performance; and (iii) the impact of the choice of the parity-check matrix is not discussed nor evaluated, despite the fact that its shape and structure has a very profound impact on the decoder's performance.

In this thesis, we present a novel message-oriented version of the model-free approach (as opposed to the codeword-oriented decoder in the literature [BCK18a; CW22a; KP20; Par+23]), which makes use of a pseudoinverse function to target only the information bits in the codeword at the expense of the parity bits. This results in a significant improvement in decoding performance, both due to the message-focused approach and the elimination of non-valid-codeword decoding outputs. Moreover, the proposed system can be straightforwardly applied to non-systematic codes, such as Polar codes in their original form [Ari09]. Finally, we study the influence of the parity-check matrix on the decoder's training and performance employing metrics from information theory [Sha48], and propose a simple algorithm to build a more suitable parity-check matrix according to these metrics.

3) Size of the neural network: r-ECCT

The third manifestation of the curse of dimensionality pertains to the number of weights (i.e., trainable parameters) in the neural network, which tends to increase drastically with the code size, and impacts particularly the model-free approach, as we will see in Chapter 3. The original work that proposed the model-free approach employed MLP and RNN [BCK18a], which involves a number of parameters that reaches 20 million for a channel chode of length $n = 127$ bits. In 2022, Choukroun et al. introduced a transformer-based architecture [CW22a] —therein referred to as Error Correction Code Transformer (ECCT)—, inspired by the work of Vaswani et al. in 2017 [Vas+17], that reduces the weight count to 2 million for the same code, while maintaining similar performances when applied to dense codes. Other works have not proposed any new deep learning-based architecture that further reduces the size of the neural network [KP20; Par+23].

In this thesis, we introduce a novel recurrent version of the ECCT, which we call r-ECCT, that tackles the space/memory constraint of model-free decoders by reducing the number of parameters to only a fraction of the parameters of the ECCT. A complexity analysis is carried out that compares the total size of each considered network as a function of the code size and other hyperparameters to be selected.

4) Extension to higher-order modulations

A final important element concerning the model-free decoding approach is that it relies heavily on the symmetry properties of the BPSK modulation scheme, where the inputs of the decoder —i.e., the syndrome and reliabilities— are independent of the transmitted codeword. This enables the single-codeword training property, which is key in the scaling capabilities of the model-free decoder. However, in order to render it applicable in realistic scenarios, the decoder must be adapted to support higher-order modulations, such as Phase-Shift Keying (PSK) and Quadrature Amplitude Modulation (QAM). As far as we can assert, no study has been carried out to analyze the application of the model-free approach to higher-order modulations, including an optimality analysis and simulation results.

In this thesis, we propose a model-free decoder that extends our previous system to the case of higher-order modulations. For this purpose, we investigate first a Bit-Interleaved Coded Modulations (BICM) scenario, which maintains the necessary symmetry properties to prove the proposed decoder's optimality. We characterize the equivalent channel between the transmitted codewords and the received Log-Likelihood Ratios (LLR) (which will serve as inputs to the decoder) using results from the literature [Alv08; CTB98]. Once the decoder's optimality is proved and the training set design is discussed, we evaluate the performance of the proposed decoder employing the main architectures from the literature (i.e., RNN and ECCT) along with the previously proposed r-ECCT. We then discuss the extension of the proposed decoder to generic Coded Modulations (CM) and illustrate our results with numerical simulations.

Organization of the work

This thesis has four chapters. Chapters 2, 3, and 4 contain our contributions and results.

Chapter 1 provides the reader with a brief overview of channel coding and linear modulations, focusing on the aspects that are necessary to understand this work. The optimal decoding rule —the so-called *Maximum A Posteriori* (MAP) decoder— is mathematically detailed and compared to the ML decoder. Finally, a few elements regarding classical decoders are presented and discussed, such as the Ordered Statistics Decoder (OSD) and the ML bound.

Chapter 2 offers an end-to-end study on SVMs and their application to channel decoding. A first theoretical deduction of maximum margin classifiers is given, followed by a state-of-the-art study on previous attempts to implement SVM-based decoders. Subsequently, a novel framework is proposed that significantly reduces the number of SVM functions to produce but also minimizes the size of the dataset needed for training. Finally, a study on complexity is carried out, the limitations of our system, and possible perspectives for future works.

Chapter 3 starts by introducing the model-free approach, its decoding framework, and its main limitations. Then, the most relevant neural-based architectures are outlined. We continue by proposing a novel *message-oriented* decoding system that significantly improves performance and is directly applicable to non-systematic codes. Additionally, a recurrent version of the transformer-based architecture is proposed, and a full complexity analysis is carried out between all the considered neural networks in terms of the total number of weights. Finally, we perform a study on the influence of the parity-check matrix on the training and performance of the decoder, proposing an algorithm to select a more convenient matrix. All studies and analyses are accompanied by their respective Bit Error Rate (BER) studies, employing several Polar and BCH codes of different sizes and rates.

Chapter 4 builds on the system proposed in Chapter 3, and extends it to higher-order modulations. For this purpose, we start by characterizing the equivalent channel between the transmitted codewords and the received LLRs in a BICM setting. Then, we prove the decoder's optimality, and discuss the difference pertaining to the training dataset compared to the BPSK-specific scenario. Finally, the decoding performance of the proposed decoder is evaluated through several short-to-medium length Polar and BCH codes.

List of publications

The work carried out throughout the thesis gave place to the following national and international publications:

International Conferences

[DB23] Gastón De Boni Rovella and Meryem Benammar. "Improved Syndrome-based Neural Decoder for Linear Block Codes." In: *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*. IEEE, Dec. 2023 (cit. on pp. 47, 54).

[DeB+24a] Gastón De Boni Rovella et al. "On the Optimality of Support Vector Machines for Channel Decoding." In: *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, June 2024 (cit. on pp. 28, 35).

[DeB+24c] Gastón De Boni Rovella et al. "Scalable Syndrome-based Neural Decoders for Bit-Interleaved Coded Modulations." In: *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. IEEE, May 2024, pp. 341–346 (cit. on p. 74).

National Conferences

[DeB+23] Gastón De Boni Rovella et al. "SVM pour la démodulation et le décodage conjoints." In: *GRETSI* (Aug. 2023) (cit. on p. 28).

Journal Articles

[DeB+24b] Gastón De Boni Rovella et al. "Optimizing the Parity-Check Matrix for Syndrome-Based Neural Decoders (**submitted**)." In: *IEEE Communications Letters* (2024) (cit. on p. 54).

[DeB+24d] Gastón De Boni Rovella et al. "Syndrome-Based Neural Decoding for Higher-Order Modulations (**submitted**)." In: *IEEE Transactions on Communications* (2024) (cit. on pp. 47, 74).

[DeB+25] Gastón De Boni Rovella et al. "Bitwise Approach for Optimal SVM-based Decoding (**submitted**)." In: *EURASIP Journal on Wireless Communications and Networking* (2025) (cit. on p. 28).

Chapter 1

Preliminaries on coding theory and linear modulations

Contents

This chapter will provide the reader with a brief introduction to coding theory, based on the work of Blaum [Bla19], along with an overview of linear modulations to be employed in this work. Then, the basic system model is presented, outlining the nomenclature employed in each stage of the communication chain. Subsequently, the Maximum Likelihood (ML) and Maximum A Posteriori (MAP) decoding rules are derived and compared. Finally, a few important classical decoders are introduced, and a procedure for computing an ML bound is described.

1.1 Introduction to binary linear block codes

1.1.1 Binary codes

Let us set the layout for the transmission of a binary frame $\boldsymbol{u} \in \{0,1\}^k$ —henceforth referred to as *message*— through a noisy channel. Due to several potential impairments (e.g., additive noise, fading, multipath, frequency or time selectivity, etc.), if we decided to transmit the desired message directly, we would very often encounter errors in reception. For this reason, channel coding is employed, where the message of length k is mapped into a longer binary frame $\boldsymbol{c} \in \{0,1\}^n$ —i.e., the *codeword*—, where $n > k$. These *extra bits* act as redundancy that allows us to retrieve the original message in case of errors inserted by the channel, or at least detect the presence of errors in the received codeword. Such a code is said to have k information bits and $n - k$ parity (or redundant) bits, and is also called a Forward Error Correction (FEC) code.

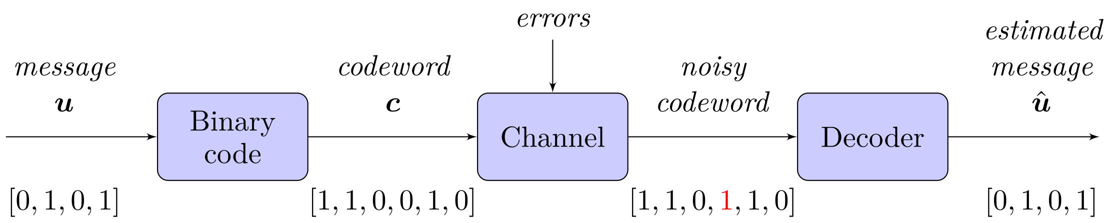


Figure 1.1: Simplified schematic of binary channel coding. An imperfect channel (e.g. a binary symmetric channel) adds errors to the transmitted codeword. The decoder seeks to correct these errors.

Definition 1.1 (Binary linear block code)**.** A binary linear block code of size (n, k) is given by a k-dimensional subspace of the n-dimensional vector space given by:

$$V_n = \{[c_1, c_2, ..., c_n] \; : \; c_i \in \mathrm{GF}(2), \; \forall i \in [1:n]\}, \tag{1.1}$$

where n is also referred to as the code *length* and k the code *dimension*, and GF(2) denotes the Galois Field of order 2.

Observe that, throughout this thesis, we work exclusively with binary linear block codes. Hence, all matrix operations, unless stated otherwise, are performed in GF(2). For an in-depth introduction to fields and group theory applied to linear codes, see [Moo05, Chapters 3 and 5].

1.1.2 Basic concepts

We have stated that a binary linear block code of length n is comprised of a set of binary sequences of length n. Consider now the following definitions regarding binary codes and their properties.

Definition 1.2 (Hamming weight)**.** Given a binary codeword $\boldsymbol{c}$ of length n where each $c_i \in \{0, 1\}$ $\forall i \in [1 : n]$, the Hamming weight $w_H(\boldsymbol{c})$ is defined as:

$$w_H(\boldsymbol{c}) \triangleq \sum_{i=1}^{n} c_i. \tag{1.2}$$

In other words, the Hamming weight $w_H(\boldsymbol{c})$ denotes the number of ones in the codeword $\boldsymbol{c}$.

Definition 1.3 (Hamming distance)**.** For two binary codewords $\boldsymbol{c}$ and $\boldsymbol{c}^*$ of length n, the Hamming distance is defined as follows:

$$d_H(\boldsymbol{c}, \boldsymbol{c}^*) \triangleq \sum_{i=1}^{n} \mathbb{1}(c_i \neq c_i^*) = \sum_{i=1}^{n} (c_i \oplus c_i^*), \tag{1.3}$$

where $\oplus$ denotes the XOR operation.

Definition 1.4 (Minimum Hamming distance)**.** Given a binary code $\mathcal{C}$, its minimum Hamming distance $d_H(\mathcal{C})$ is given by the shortest Hamming distance between any two codewords. That is:

$$d_H(\mathcal{C}) \triangleq \min_{\boldsymbol{c}, \boldsymbol{c}^* \in \mathcal{C}} d_H(\boldsymbol{c}, \boldsymbol{c}^*). \tag{1.4}$$

Definition 1.5 (Equivalent codes)**.** Two linear block codes over GF(2) —i.e. binary codes— are equivalent if one can be obtained from the other by applying permutations of the indices within the codewords.

Example 1.1 (Equivalent codes)**.** Suppose we have the following linear code of length 3:

$$\mathcal{C} : \{[0, 0, 0], [1, 1, 0], [1, 0, 1]\}. \tag{1.5}$$

Then, we can easily verify that is minimum Hamming distance is 2 and that the code $\mathcal{C}^*$ given by:

$$\mathcal{C}^* : \{[0, 0, 0], [0, 1, 1], [1, 0, 1]\} \tag{1.6}$$

is equivalent to $\mathcal{C}$, where the first and third bits have been permuted with respect to $\mathcal{C}$.

Observe that a larger minimum Hamming distance leads to a code where its codewords are further away from each other, thus increasing the error-correcting capabilities of the code. Observe also that equivalent codes possess the same minimum Hamming distance.

1.1.3 Generator and parity-check matrices

A binary linear block code $\mathcal{C}$ can be characterized by a binary matrix of size $k \times n$, called the *generator matrix*. For a $(7, 4)$ Hamming code [RL09] —i.e., $n = 7$ and $k = 4$—, a possible generator matrix is expressed as follows:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{1.7}$$

and given a message $\boldsymbol{u} \in \{0,1\}^k$, its corresponding codeword $\boldsymbol{c} \in \{0,1\}^n$ can be obtained by applying the product in GF(2):

$$\boldsymbol{c} = G^T \boldsymbol{u}, \tag{1.8}$$

where both $\boldsymbol{c}$ and $\boldsymbol{u}$ are column vectors. A codeword obtained following this procedure is called a *valid* codeword, and all together, they constitute the code $\mathcal{C}$, composed of 2^k codewords of length n (one for each possible message $\boldsymbol{u} \in \{0,1\}^k$). Observe that a code consists of the entire space generated by the rows of the generator matrix in GF(2). Conversely, a code is also defined by its *parity-check matrix*. A matrix H of size $(n-k)\times n$ is a parity-check matrix of the code $\mathcal{C}$ if it satisfies, $\forall\, \boldsymbol{c} \in \{0,1\}^n$:

$$H\boldsymbol{c} = \mathbf{0}_{n-k} \;\Leftrightarrow\; \boldsymbol{c} \in \mathcal{C}, \tag{1.9}$$

where $\mathbf{0}_{n-k}$ denotes the all-zero vector of length $n-k$. Analogously, a binary frame $\tilde{\boldsymbol{c}} \in \{0,1\}^n$ such that $H\tilde{\boldsymbol{c}} \neq \mathbf{0}_{n-k}$ is denominated a *non-valid* codeword, and is often the result of a transmission over a noisy channel. For the previous Hamming code, a possible parity-check matrix is given by:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1.10}$$

Finally, observe that a pair of matrices G (of size $k\times n$) and H (of size $(n-k)\times n$) of full row rank constitute a valid pair of generator and parity-check matrices if and only if they verify:

$$GH^T = \mathbf{0}_{k,n-k}, \tag{1.11}$$

where $\mathbf{0}_{k,n-k}$ is the all-zero matrix of size $k \times (n-k)$.

Consider now the following theorem from [Hil90, Chapter 5], applied particularly to binary linear block codes.

Theorem 1.1 (Generator matrices of equivalent codes)**.** *Two $k\times n$ matrices generate equivalent codes over GF(2) if one matrix can be obtained from the other by a sequence of operations of the following types:*

(i) Permutation of the rows.

(ii) Addition of one row to another.

(iii) Permutation of the columns.

Proof. The proof can be found in [Hil90, Chapter 5]. □

1.1.4 Matrix manipulations

Given a generator matrix G of a linear code $\mathcal{C}$, it can be easily verified that any matrix $\tilde{G}$ of full row rank obtained by applying permutations and combinations of the rows of G is also a generator matrix for the code $\mathcal{C}$. The same is valid for the parity-check matrix H. Given this observation, let us introduce the following definition.

Definition 1.6 (Systematic and standard matrices)**.** A matrix A of size $k \times n$ is said to be in systematic form if and only if it can be expressed as:

$$A_s = [\, I_k \,|\, P \,], \tag{1.12}$$

by only applying column permutations, where I_k is the identity matrix of size k, P designates a $k \times (n-k)$ matrix and A_s is the permuted version of A. If no permutations are needed to obtain the form in (1.12), then the matrix is said to be in its standard form.

When we have access to the generator matrix in its standard form $G_s = [\, I_k \,|\, P \,]$, the parity-check matrix can be easily computed as:

$$H_s = [\, P^T \,|\, I_{n-k} \,], \tag{1.13}$$

and similarly from a standard parity-check matrix to its corresponding generator matrix. A simple method for standardizing a generator matrix can be found in [Hil90, Chapter 5]. For the previous $(7,4)$ Hamming code, the standard generator matrix and its corresponding parity-check matrix are expressed as follows:

$$G_s = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad H_s = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.14}$$

1.1.5 Polar and BCH codes

In this thesis, the two main codes that will be used are the denominated Polar codes [Ari09] and Bose–Chaudhuri– Hocquenghem (BCH) codes [BR60; Hoc59]. Let us begin with a brief introduction to Polar codes. BCH codes will not be discussed in depth but references are provided in the end for interested readers.

Let $P_n = F^{\otimes \log_2 n}$, where n is a power of 2, $F^{\otimes i}$ represents the ith Kronecker power of F, and F denotes Arikan's kernel, given by:

$$F \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{1.15}$$

Observe that P_n is not defined if n is not a power of 2. A generator matrix for a Polar code of size (n, k) is given by concatenating k rows of the matrix P_n. How to choose these k rows is a research field in and of itself and will not be covered in this short introduction, but for more information, the reader is referred to [Pfi17; TV13]. Observe that, as per the definition of P_n, n must be a power of 2.

Lemma 1.1 (Parity-check matrix of a Polar code)**.** *Let G denote a generator matrix for a Polar code of size (n, k), composed of k rows of the matrix P_n, and let $\mathcal{A} \subset \{1, 2, ..., n\}$ denote the indices of these rows. Then, the matrix H of size $(n-k) \times n$ consisting of the columns of P_n with indices in the complement set of $\mathcal{A}$, i.e., $\mathcal{A}^c$, is a valid parity-check matrix for the code defined by G.*

Proof. The lemma can be easily proven by considering that the matrix product GH^T consists of dot products between a row and a column of P_n with different indices, which is always 0 because P_n is, by construction, an involutory matrix (i.e., $P_nP_n = I_n$). □

Example 1.2 (Polar code of size $(8, 4)$)**.** Consider a case where we wish to construct a Polar code with $k = 4$ information bits and $n = 8$ total bits. Then, the base matrix is given by:

$$P_8 = F^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1.16}$$

There are several methods to select the rows from P_8, such as the Bhattacharyya construction (as it is based on Bhattacharyya bounds for computing the error probability) proposed by Arikan in his original work [Ari09] or the improved degrading-merge algorithm from [TV13] by Tal and Vardy. Using the Bhattacharyya construction, the resulting generator matrix is the following:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1.17}$$

Finally, using Lemma 1.1, we obtain the following parity-check matrix:

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1.18}$$

This encoding procedure is traditionally displayed as in Figure 1.2, where the channels used to transmit the information bits are (u_4, u_6, u_7, u_8), and the remaining channels are called *frozen bits* and are commonly set to 0. This diagram is also called a *factor graph*.

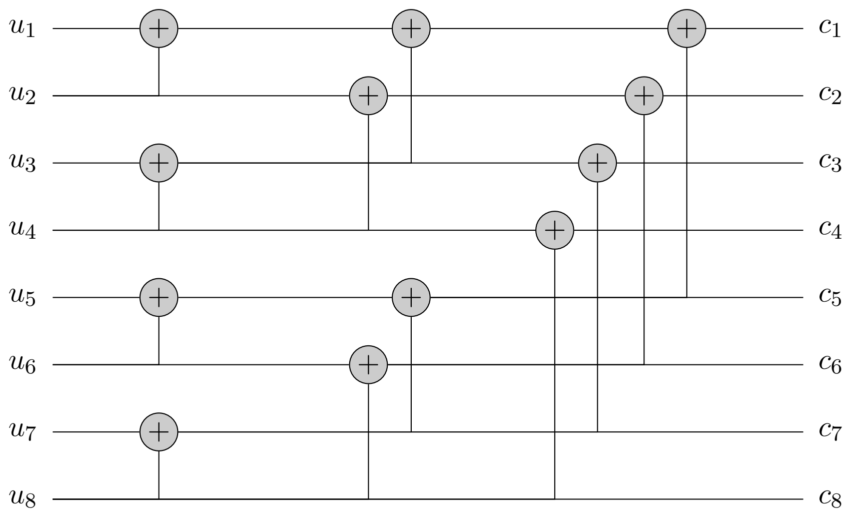


Figure 1.2: Polar encoder diagram (or factor graph) of Example 1.2.

BCH codes, on the other hand, are a class of error-correction codes mainly used in satellite communications and data storage devices, and their major advantage lies in the precise control over the number of codeword errors that can be corrected by the code. The code length is given by $n = 2^m - 1$, where m is the degree of the finite field used. In this scenario, a BCH code can correct up to t errors, where:

$$n - k \geq mt. \tag{1.19}$$

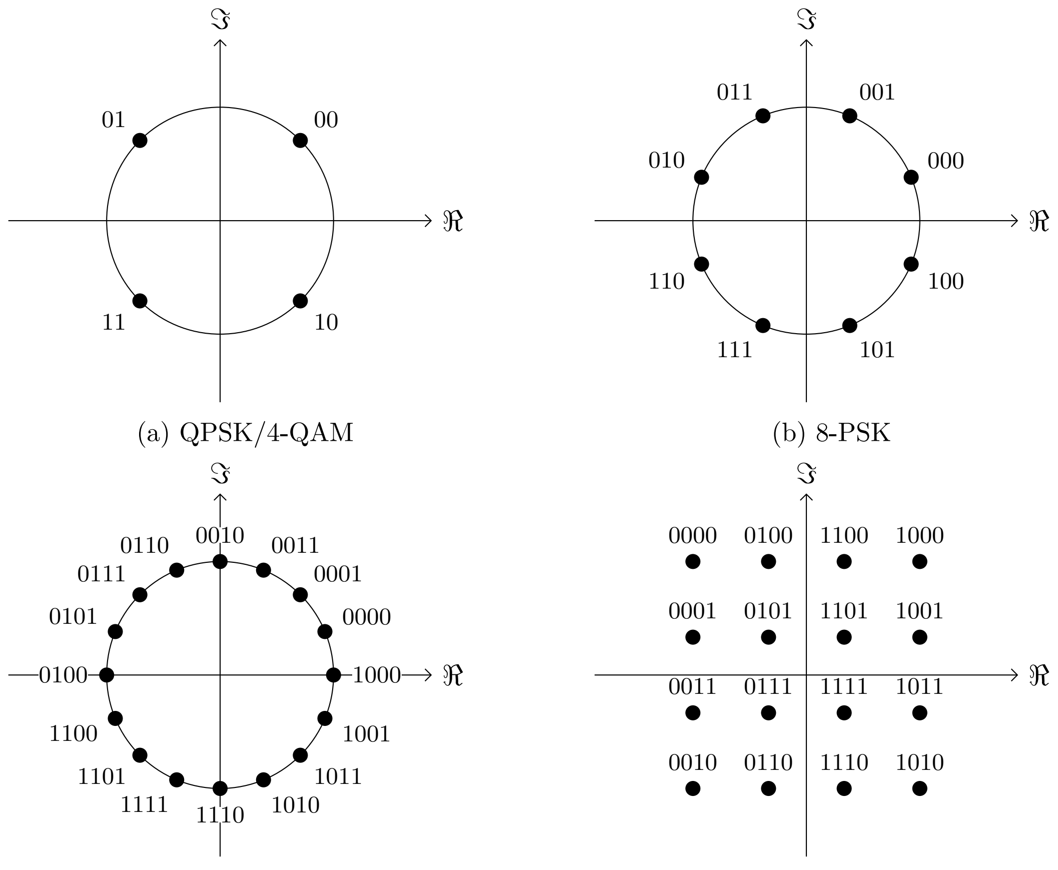
This allows us to select the number of parity bits necessary to achieve the desired error correction capability. In their construction, BCH codes are characterized by the roots in the so-called *generator polynomials*. For an in-depth tutorial on their construction and properties, the reader is referred to [RL09, Section 3.3].

1.2 Linear modulations and log-likelihood ratios

Once the channel code to be employed is selected, the binary codeword has to be converted into a complex symbol before being transmitted through the channel. This process is known as *modulation*, and given an order m, assigns every possible combination of m bits to a point in the complex plane. In this work, we will use the Binary Phase-Shift Keying (BSPK) modulation of order $m = 1$, defined as follows:

$$x_{\text{BPSK}} = \begin{cases} -1 & \text{if } u = 1 \\ +1 & \text{if } u = 0, \end{cases} \tag{1.20}$$

along with other higher-order modulations displayed in Figure 1.3 —the Quadradature Amplitude Modulation of order m (2^m-QAM) and the Phase-Shift Keying of order m (2^m-PSK). Observe that Gray labeling is employed so that adjacent symbols differ in, at most, one bit.



(a) QPSK/4-QAM (b) 8-PSK (c) 16-PSK (d) 16-QAM

Figure 1.3: Constellation diagrams.

On the receiving side, the demodulator computes the so-called *Log-Likelihood Ratio* (LLR) for each transmitted bit, defined for every bit position $i \in [1:n]$ as:

$$l_i \triangleq \log\left(P_{Y|C_i}(y|1)\right) - \log\left(P_{Y|C_i}(y|0)\right), \tag{1.21}$$

where C_i and Y denote respectively the random variables of the ith bit of the codeword $\boldsymbol{c} \in \{0,1\}^n$ and its resulting noisy symbol $y \in \mathbb{C}$ in reception (corresponding to c_i and $m-1$ other bits), and $P_{Y|C_i}$ denotes their associated conditional probability density function (pdf).

Example 1.3 (BPSK and AWGN)**.** Consider the following simple case: a uniform source of independent and identically distributed (iid) bits generates frames $\boldsymbol{u} \in \{0,1\}^k$. Then, a linear encoder maps this message into a codeword $\boldsymbol{c} \in \{0,1\}^n$, and after a BPSK modulation, we obtain the signal to be transmitted $\boldsymbol{x} \in \{-1,+1\}^n$. The channel inserts a real Additive

White Gaussian Noise (AWGN) of power σ^2, described by the following pdf:

$$f_{\text{AWGN}}(w) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{w^2}{2\sigma^2}}. \tag{1.22}$$

In this case, the LLR boils down to the following expression:

$$l_i = \log \frac{e^{-\frac{(y+1)^2}{2\sigma^2}}}{e^{-\frac{(y-1)^2}{2\sigma^2}}} = -\frac{2y}{\sigma^2}. \tag{1.23}$$

Intuitively, a positive value of l_i indicates a higher probability for the ith bit to be equal to 0, whereas a negative value corresponds to a higher likelihood of a 1. Observe that l_i close to 0 implies the largest possible uncertainty on the bit's actual value.

1.3 System model

Let us introduce the base system model we will use throughout this work. Consider the communication layout of Figure 1.4. The binary frame $\boldsymbol{u} \in \{0,1\}^k$ is encoded through a linear FEC code $\mathcal{C}$ defined by its generator matrix G, such that $\boldsymbol{c} = G^T\boldsymbol{u}$. This codeword is then modulated using a linear modulation technique of order m —such as QAM or PSK—, which produces the complex signal to be transmitted $\boldsymbol{x} \in \mathbb{C}^{n'}$, where $n' = n/m$. The channel adds a complex and circular AWGN $\boldsymbol{w} \sim \mathcal{CN}(\boldsymbol{0}, I_{n'}\sigma^2)$, such that $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{w}$. On reception, the demodulator computes the LLR vector $\boldsymbol{l}$ as in (1.21), and finally, the decoder $\boldsymbol{g}(\cdot)$ outputs an estimate $\hat{\boldsymbol{u}}$ of the originally transmitted message:

$$\hat{\boldsymbol{u}} = \boldsymbol{g}(\boldsymbol{l}). \tag{1.24}$$

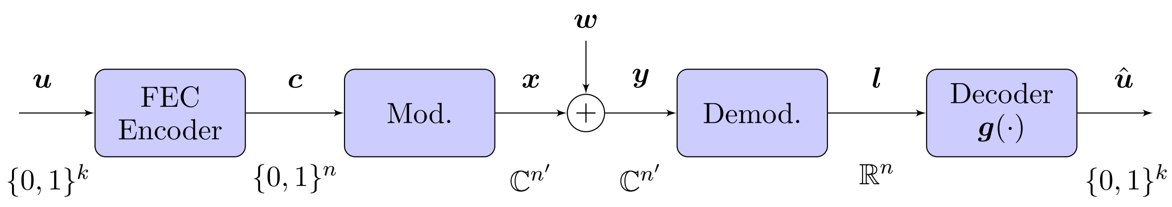


Figure 1.4: System model of a simplified communication chain. For clarity, the domain of the signal in each stage of the system is added below.

1.3.1 The decoding problem: MAP and bit-MAP

The so-called *decoding problem* consists in producing a function $\boldsymbol{g}(\cdot)$ that minimizes the probability of error at the receiver. Observe that the input signal to the function $\boldsymbol{g}(\cdot)$ can be the LLR vector $\boldsymbol{l}$ (in the case separate demodulation and decoding) or the channel output $\boldsymbol{y}$ (in the case of joint demodulation and decoding). In what follows, let us assume the latter scenario.

The error probability can be measured in two ways: (i) comparing the transmitted and decoded frames as a whole, considering a *frame error* if at least one bit in the frame is decoded incorrectly, and (ii) comparing each transmitted bit with its corresponding decoded bit. If the probability of error is measured framewise, that is:

$$\overline{P_e} = \mathbb{P}\{\boldsymbol{u} \neq \hat{\boldsymbol{u}}\}, \tag{1.25}$$

where $\overline{P_e}$ denotes the Frame Error Probability (FEP), then the optimal decoder is known as the MAP decoder and can be expressed as:

$$\boldsymbol{g}(\boldsymbol{y}) \triangleq \underset{\boldsymbol{u}\in\{0,1\}^k}{\operatorname{argmax}}\ P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}). \tag{1.26}$$

Conversely, if the error probability is measured bitwise, i.e.:

$$P_e = \frac{1}{k}\sum_{i=1}^{k} \mathbb{P}\{u_i \neq \hat{u}_i\}, \tag{1.27}$$

where P_e denotes the Bit Error Probability (BEP), then the optimal decoder is known as the bit-MAP decoder, and is defined for every $i \in [1:k]$ as:

$$g^{(i)}(\boldsymbol{y}) \triangleq \underset{u\in\{0,1\}}{\operatorname{argmax}}\ P_{U_i|\boldsymbol{Y}}(u|\boldsymbol{y}). \tag{1.28}$$

1.3.2 MAP vs. ML

In this work, we will often use uniform binary sources, where every binary frame is equiprobable. In this case, we can rewrite the MAP probability expression using Bayes' formula:

$$\begin{aligned}\underset{\boldsymbol{u}\in\{0,1\}^k}{\operatorname{argmax}} P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) &= \underset{\boldsymbol{u}\in\{0,1\}^k}{\operatorname{argmax}} \frac{P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u})P_{\boldsymbol{U}}(\boldsymbol{u})}{P_{\boldsymbol{Y}}(\boldsymbol{y})} &(1.29)\\ &= \underset{\boldsymbol{u}\in\{0,1\}^k}{\operatorname{argmax}} \frac{P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u})}{P_{\boldsymbol{Y}}(\boldsymbol{y})2^k} &(1.30)\\ &= \underset{\boldsymbol{u}\in\{0,1\}^k}{\operatorname{argmax}} P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u}), &(1.31)\end{aligned}$$

where we have used that $P_{\boldsymbol{U}}(\boldsymbol{u}) = 1/2^k$, along with the fact that $P_{\boldsymbol{Y}}(\boldsymbol{y})$ is independent from $\boldsymbol{u}$ and thus $P_{\boldsymbol{Y}}(\boldsymbol{y})$ can be removed from the argmax expression. The expression in (1.31) is known as the ML criterion. In summary, when all the messages are equiprobable, the ML criterion coincides with the MAP. By applying similar reasoning, we can derive the corresponding result for bitwise decoding for $i \in [1:k]$:

$$
\begin{aligned}
g^{(i)}(\boldsymbol{y}) &\triangleq \underset{u\in\{0,1\}}{\operatorname{argmax}}\; P_{U_i|\boldsymbol{Y}}(u|\boldsymbol{y}) && (1.32)\\
&= \underset{u\in\{0,1\}}{\operatorname{argmax}} \sum_{\substack{\boldsymbol{u}\in\{0,1\}^k\\ /u_i=u}} P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) && (1.33)\\
&= \underset{u\in\{0,1\}}{\operatorname{argmax}} \sum_{\substack{\boldsymbol{u}\in\{0,1\}^k\\ /u_i=u}} \frac{P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u})P_{\boldsymbol{U}}(\boldsymbol{u})}{P_{\boldsymbol{Y}}(\boldsymbol{y})} && (1.34)\\
&= \underset{u\in\{0,1\}}{\operatorname{argmax}} \frac{2^{-k}}{P_{\boldsymbol{Y}}(\boldsymbol{y})} \sum_{\substack{\boldsymbol{u}\in\{0,1\}^k\\ /u_i=u}} P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u}) && (1.35)\\
&= \mathbb{1}\Bigg\{ \sum_{\substack{\boldsymbol{u}\in\{0,1\}^k\\ /u_i=1}} P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u}) > \sum_{\substack{\boldsymbol{u}\in\{0,1\}^k\\ /u_i=0}} P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u}) \Bigg\}, && (1.36)
\end{aligned}
$$

where we have used that the source is uniform, i.e., $P_{\boldsymbol{U}}(\boldsymbol{u}) = 2^{-k}$ for all $u \in \{0,1\}^k$, and we have discarded from the argmax the factors that do not depend on the optimizing variable u.

Observation 1.1. As previously stated, when considering separate demodulation and decoding, the input to the decoder is the LLR vector $\boldsymbol{l}$. In this scenario, we can derive the expression that maximizes the probabilities $P_{\boldsymbol{U}|\boldsymbol{L}}(\boldsymbol{u}|\boldsymbol{l})$ (MAP decoding) and $P_{U_i|\boldsymbol{L}}(u|\boldsymbol{l})$ $\forall i \in [1:n]$ (bit-MAP decoding), which do not necessarily coincide with the optimal decoders that take $\boldsymbol{y}$ as their input. As we shall see later on, Chapters 2 and 3 employ joint demodulation and decoding, while Chapter 4 implements them separately.

1.4 Classical decoders

Throughout this thesis, classical decoding algorithms are employed for two main purposes: (i) to estimate a pseudo-optimal decoder in cases where the MAP decoder is too complex to be implemented and (ii) to compare our proposed solutions and those in the literature to those used in realistic industrial applications. For the former reason, a very useful algorithm is the Ordered Statistics Decoder (OSD), introduced by Fossorier and Lin in 1995 [FL95]. A short description of the decoding procedure is given below.

1.4.1 Ordered Statistics Decoder

Let us suppose a transmission scenario as the one described in Section 1.3 and Figure 1.4, where an (n,k) binary linear code $\mathcal{C}$ is employed, and is described by a generator matrix G. On reception, the demodulator computes the LLR vector $\boldsymbol{l} = (l_1, l_2, ..., l_n)$. The OSD consists of two stages: the ordered statistics preprocessing and the reprocessing stages.

Ordered statistics preprocessing

This stage begins by ordering the vector of log-likelihood ratios $\boldsymbol{l}$ in decreasing order of reliability value, i.e.:

$$\boldsymbol{l}' = (l'_1, l'_2, ..., l'_n), \tag{1.37}$$

such that $|l'_1| > |l'_2| > ... > |l'_n|$. If we denote this permutation λ_1, we can apply λ_1 to G to obtain a generator matrix G' which defines a code $\mathcal{C}'$ equivalent to $\mathcal{C}$ as per definition 1.5. That is:

$$G' = \lambda_1[G]. \tag{1.38}$$

Starting from the leftmost column of this new generator matrix G', we find the first k linearly independent columns, associated with the largest possible reliabilities. The bits corresponding to these positions are denoted as the Most Reliable Independent (MRI) positions. This constitutes a second permutation λ_2, which in turn defines a third generator matrix:

$$G'' = \lambda_2[G'] = \lambda_2[\lambda_1[G]]. \tag{1.39}$$

The same permutation is applied to $\boldsymbol{l}'$ to obtain the MRI bits grouped to the left of the codeword:

$$\boldsymbol{l}'' = \lambda_2[\boldsymbol{l}'] = \lambda_2[\lambda_1[\boldsymbol{l}]] = (l''_1, l''_2, ..., l''_n). \tag{1.40}$$

Next, row-wise elementary operations are applied on G'' in order to obtain a left-diagonal matrix, which is possible since, as a result of the second permutation λ_2, the k leftmost columns are linearly independent. This results in a matrix G_d such that:

$$G_d = [\, I_k \,|\, P \,], \tag{1.41}$$

where I_k is the identity matrix of size k and P is a $k \times (n-k)$ parity matrix. Once we reach this stage, the first k elements of the vector $\boldsymbol{l}''$ are kept, and the remaining $n-k$ elements are discarded. Let us call this vector $\bar{\boldsymbol{z}}$, which is then used to generate a codeword $\boldsymbol{z}$ from the diagonal generator matrix G_d, i.e.,

$$\boldsymbol{z} = G_d^T \bar{\boldsymbol{z}}. \tag{1.42}$$

Once we have computed $\boldsymbol{z}$, which belongs to the code $\mathcal{C}''$, we can return to the original code $\mathcal{C}$ by applying the inverse permutations as before:

$$\hat{\boldsymbol{c}} = \lambda_1^{-1}[\lambda_2^{-1}[\boldsymbol{z}]], \tag{1.43}$$

where $\hat{\boldsymbol{c}}$ is an estimate of the original codeword $\boldsymbol{c}$ that relies on the MRI positions of the received LLR vector $\boldsymbol{l}$.

Reprocessing

The next stage of the algorithm consists essentially in testing a certain number of changes in the MRI bits and verifying which resulting codeword has a higher probability.

The *order-0* reprocessing stage is the algorithm explained above, as no bit in $\bar{\boldsymbol{z}}$ is flipped. To begin the *order-1* reprocessing stage, we change the decision of the first MRI bit —i.e., $\bar{z}_1$— and use the modified binary frame to compute $\boldsymbol{z}$, which in turn is used to obtain a new estimate $\hat{\boldsymbol{c}}^*$. Next, we have to compare the probability of having transmitted $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}^*$. If we are dealing with BPSK signals, then the demodulator would not be necessary, and we can just take the absolute value of the received signal $\boldsymbol{y}$ as measure for the reliabilities. On a generic case, we must compute the Euclidean distance between the received signal $\boldsymbol{y}$ and the estimated codewords $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}^*$ modulated into the appropriate complex signal and select the one that has the smallest distance:

$$P(\hat{\boldsymbol{c}}|\boldsymbol{y}) > P(\hat{\boldsymbol{c}}^*|\boldsymbol{y}) \iff ||\text{mod}(\hat{\boldsymbol{c}}) - \boldsymbol{y}||^2 < ||\text{mod}(\hat{\boldsymbol{c}}^*) - \boldsymbol{y}||^2, \tag{1.44}$$

where mod$(\cdot)$ is a function that maps the codewords $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}^*$ into their respective modulated signals. In our work, to simplify the handling of higher-order modulations, this probability is computed using the LLR-based distance from [BBB15], defined as follows:

$$d(\boldsymbol{l}, \hat{\boldsymbol{c}}) \triangleq \sum_{i=1}^{n} \log(1 + e^{-(1-2\hat{c}_i)l_i}), \tag{1.45}$$

where $\boldsymbol{l}$ and $\hat{\boldsymbol{c}}$ denote the LLR vector on reception and the estimated codeword, respectively. Using this distance metric, we decide between the two estimated codewords $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}^*$ and keep the most probable one. We continue this reasoning for the other $k-1$ bits in $\bar{\boldsymbol{z}}$ to finish the order-1 OSD.

Higher orders are implemented in the same manner, flipping as many bits as the current order of reprocessing. When employing an OSD, we begin by selecting its order, which indicates how many stages of reprocessing will be employed. For an order-p OSD, the total number of computations is given by:

$$\sum_{i=0}^{p} \binom{k}{i}. \tag{1.46}$$

The OSD will be mainly used for decoding BCH codes since the ML decoder is intractable for the code lengths we employ. As proved in [FL95, Section V.E], the OSD converges asymptotically to the ML decoder provided that the order l satisfies the following inequality:

$$p \geq \min\{\lceil d_H/4 - 1\rceil, k\}, \tag{1.47}$$

where d_H denotes the code's minimum Hamming distance. In our experimental scenarios, an order of 2 or 3 will be enough to offer performances very close to the optimal ML decoder.

1.4.2 Successive Cancellation decoder

The Successive Cancellation (SC) decoder was proposed by Arikan in his original work on Polar codes, along with its associated performance bounds. In this work, we will provide a short description of the algorithm. For further details, the reader is referred to [Ari09; Pfi17].

The procedure detailed in Section 1.1.5, defines a method for constructing codes by channel polarization —hence its name—, which transforms n independent copies of a binary-input discrete memoryless channel into n bit channels with varying reliabilities. The channels with the highest reliabilities are employed to transmit the information bits. The SC decoding algorithm decodes each bit of the received vector sequentially, using previously decoded bits to assist in the decision-making process for subsequent bits. SC decoding employs a recursive approach to handle the polarization structure of the code. The algorithm can be described in terms of a binary tree, where each node represents a combination or split operation corresponding to the matrix, as defined by Arikan's kernel F given in (1.15):

(i) The decoding starts with the received vector $\boldsymbol{l}$, which consists of the LLRs from the channel, which are assigned to the right side of Figure 1.2.

(ii) Applying the appropriate mathematical formulas, the likelihoods are combined and transmitted through the factor graph, until they reach the first bit u_1, where the value is either known or unknown.

(iii) If it is unknown (unfrozen bit), then a decision on the value of $\hat{u}_1$ is made. Otherwise, the knowledge of the value of the frozen bit $u_1 = 0$ is used to update the likelihoods.

(iv) This likelihood transmission continues until it reaches the second bit u_2, and the same reasoning is applied.

(v) This procedure is followed sequentially, exploiting for each bit u_i the LLRs updated with the hard decisions on the bits $u_1, ..., u_{i-1}$, until all the bits have been decoded.

In this work, we will also make use of the Successive Cancellation List (SCL) decoding, proposed by Tal and Vardy in 2011 [TV11], which involves storing the L most likely outcomes —or *paths*— and deciding between them according to their final probability. This technique is much more complex, and its algorithm is not included in this work. However, it is the base decoder for Polar codes (e.g., in the control channel of the 5G New Radio protocol of the 3GPP [3GP20; BCL21]), and thus will be employed as a benchmark in this work.

Table 1.1 summarizes the computational and memory complexity of the three classical decoders previously presented.

1.4.3 Maximum Likelihood bound

The following Maximum Likelihood Bound (MLB), described in [TV11], is a very useful tool that will be extensively employed throughout this work. This bound allows us to evaluate

Decoding method	Computational complexity	Memory complexity
SC	$\mathcal{O}(n\log(n))$	$\mathcal{O}(n)$
SCL	$\mathcal{O}(Ln\log(n))$	$\mathcal{O}(Ln)$
OSD	$\mathcal{O}(2^p n^2)$	$\mathcal{O}(n^2)$

Table 1.1: Complexity of the classical decoding algorithms, where n denotes the code length, L the list size of the SCL and p the order of the OSD.

if the simulated solutions approach the decoding performance of the optimal ML decoder. Additionally, when an implemented decoder (e.g., an OSD of order 2) matches the MLB, then we can say with certainty that the decoder attains ML performance.

Figure 1.5 shows a flowgraph that details the MLB procedure. Consider the communication scenario of Section 1.3, but with a decoder that outputs $\hat{\boldsymbol{c}}$, an estimate of the codeword $\boldsymbol{c}$ (observe that if the decoding algorithm produces uniquely an estimate $\hat{\boldsymbol{u}}$, we can easily compute $\hat{\boldsymbol{c}} = G^T\hat{\boldsymbol{u}}$). Then, we must be in one of two situations: (i) our decoder estimated correctly (i.e., $\hat{\boldsymbol{c}} = \boldsymbol{c}$), in which case we will consider that the ML decoder would have also decoded correctly; or (ii) our decoder incurred an error (i.e., $\hat{\boldsymbol{c}} \neq \boldsymbol{c}$). In the latter case, we compute the likelihood $P_{\boldsymbol{L}|\boldsymbol{C}}(\boldsymbol{l}|\hat{\boldsymbol{c}})$ of the estimated codeword $\hat{\boldsymbol{c}}$, and if it is larger than the likelihood of $\boldsymbol{c}$, then a true ML decoder would have decoded incorrectly as well. This is the only case we will consider a decoding error for the MLB.

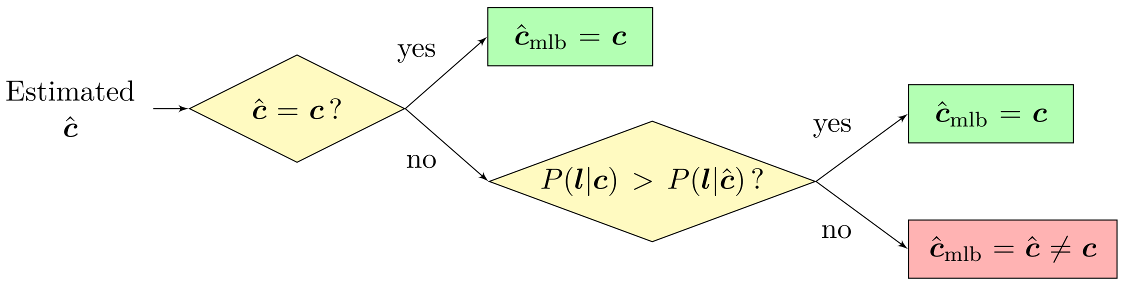

Estimated $\hat{\boldsymbol{c}}$
$\hat{\boldsymbol{c}} = \boldsymbol{c}\,?$
yes
no
$\hat{\boldsymbol{c}}_{\text{mlb}} = \boldsymbol{c}$
$P(\boldsymbol{l}|\boldsymbol{c}) > P(\boldsymbol{l}|\hat{\boldsymbol{c}})\,?$
yes
no
$\hat{\boldsymbol{c}}_{\text{mlb}} = \boldsymbol{c}$
$\hat{\boldsymbol{c}}_{\text{mlb}} = \hat{\boldsymbol{c}} \neq \boldsymbol{c}$

Figure 1.5: Maximum likelihood bound flowchart. $\hat{\boldsymbol{c}}_{\text{mlb}}$ denotes the MLB-estimated codeword.

Observation 1.2. It is easy to see that the better the base decoder used to compute the bound is, the tighter the bound will be. For this reason, we will often make use of a high-order OSD to produce the MLB curve. Moreover, when the FER of an implemented decoder coincides with its corresponding MLB, we are certain that our decoder achieves the optimal ML performance.

Observation 1.3. This bound is well-suited for a frame-wise approach, since when a decoding error is detected, we can only say with certainty that the estimated codeword $\hat{\boldsymbol{c}}$ has a higher probability than $\boldsymbol{c}$, but we cannot know if it constitutes *the most* probable codeword. However, it is very unlikely that the decoder estimated a codeword that has a higher probability than the transmitted one when there exists a third one with an even higher probability. For this reason, we will make use of this bound for the BER along with the FER.

Chapter 2

Support Vector Machines for joint demodulation and decoding

Contents

As stated in the Introduction, the curse of dimensionality we tackle in this work manifests itself in different ways. One of them pertains to the number of noise realizations: for each possible output, a generic deep learning-based decoder must explore several input patterns that yield that same output. That is, for each valid codeword, a neural network must *see* different noisy realizations of that codeword to *learn* to decode it under a random noise. This calls for another machine learning algorithm that achieves successful learning with a reduced dataset.

Support Vector Machines (SVM) constitute a supervised learning method that, from a labeled dataset with two classes, produces a classifying decision rule that is at an equal and maximum distance from both classes. For this reason, it is known as a *maximum margin* classifier and is well-known for its robustness and interpretability. This optimal decision rule is generated using the so-called *support vectors*, which usually constitute a small subset of training samples. Inspired by this notion, we study SVMs as a potential decoding algorithm that reduces the minimum size of the training dataset and is robust to mismatches in training and testing conditions.

Previous works have already tackled the problem of SVM for channel coding [DH10; KB08; SY16]. The work in [DH10] employs SVMs for online adaptive modulation and coding, displaying favorable results with reduced complexity compared to previous algorithms. The authors in [KB08] proposed a novel pairwise SVM-based decoder that achieves competitive performances on convolutional codes, albeit at a very high complexity. The same analysis applies to the works in [Ram+09] and [SY16], which propose small modifications in training and application scenarios but keep the pairwise classification approach. For a channel code of length n and dimension k, this approach has two main limitations:

(i) The dataset, which is composed of several noisy realizations of every valid codeword, grows exponentially with k (as the codeword space expands) and with n (as the number of correctable noise patterns increases);

(ii) The decoder contains a number of binary classifiers that also increases exponentially with k (due to the nature of the pairwise approach).

These two aspects result in an intractable decoder when applied to a code that is longer than a few bits, usually a Hamming code of size $(7, 4)$. The largest implemented code is a BCH of size $(15, 7)$ in [SY16], and employs up to 100 noisy realizations of each valid codeword.

This Chapter describes in detail how SVMs can be used for channel decoding, and is organized as follows. We start by providing the reader with a brief introduction to SVMs, inspired by [AML12], followed by a state-of-the-art study on previous SVM applications to channel decoding. Subsequently, we propose a new system and training framework that reduces the number of SVMs to produce from 2^k to only k, and reduces the training dataset to its minimum. Finally, a mathematical analysis is provided to establish the equivalence between the proposed SVM decoder and the bit-ML decoder under AWGN conditions, followed by a complexity study and a few perspectives on the subject.

The integrality of this work was published in a national conference [DeB+23] and an international conference [DeB+24a]. Additionally, a journal article is currently under review [DeB+25].

2.1 Introduction to SVMs

SVMs were introduced over thirty years ago by Vapnik, Boser, Cortes, and Guyon [BGV92; CV95; Vap97]. The main idea consists in using a set of labeled data —each element belonging to one of two possible classes— to produce a separating hyperplane that divides the data into the two corresponding classes whilst maximizing the margin between the hyperplane and the two classes. For this reason, it is known as a *maximum margin classifier*. In the following, a brief introduction to SVMs is given, along with the necessary elements to understand our proposed SVM-based receiver —i.e., the kernel method and the multiclass SVM.

2.1.1 System model and preprocessing

SVMs are, by nature, maximum margin binary classifiers. Given a fully labeled dataset with two disjoint classes (i.e., no element belongs to both classes), the SVM classifier is given by the hyperplane that separates the two classes, and that has the largest possible margin between them and the hyperplane. This section will employ SVMs for joint channel demodulation and decoding. As such, and because SVMs work on real numbers, the received complex signal $\boldsymbol{y}$ is preprocessed before being inputted to the SVM decoder (see Figure 2.1):

$$\tilde{\boldsymbol{y}} = [\,\mathcal{R}(\boldsymbol{y}), \mathcal{I}(\boldsymbol{y})\,]. \tag{2.1}$$

This vector $\tilde{\boldsymbol{y}}$ will be the input to the SVM-based receiver, and the output will be an estimation of the message, $\hat{\boldsymbol{u}}$. Also, observe that demodulation and decoding is performed jointly by the SVM decoder, which receives directly the channel output without previous demodulation. As such, this Chapter will not deal with LLRs. The remaining elements of the communication layout are exactly the same as in Section 1.3.

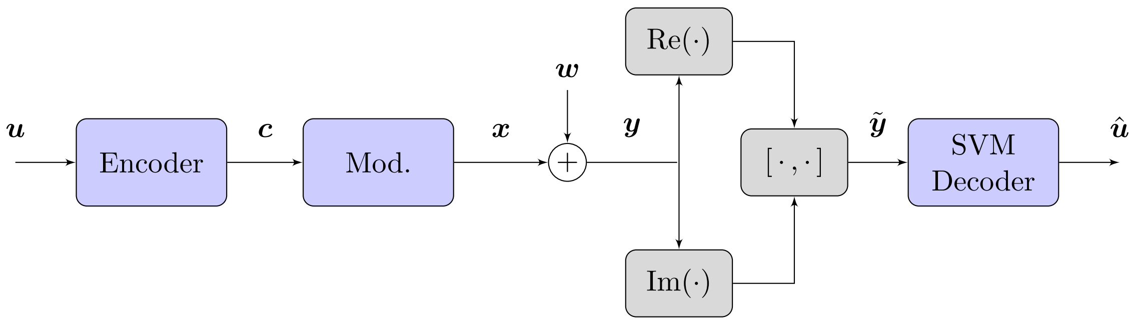


Figure 2.1: System model of a simplified communication chain, including the concatenation of the real and imaginary parts for the SVM processing.

2.1.2 Linearly separable data

Let us consider a labeled dataset $\{\tilde{\boldsymbol{y}}_i, l_i\}_{1\leq i\leq N}$ which consists of N vectors $\tilde{\boldsymbol{y}}_i \in \mathbb{R}^{2n'}$ with their respective labels $l_i \in \{-1, +1\}$[1]. Let the class $\mathcal{C}_0$ be the set of vectors $\tilde{\boldsymbol{y}}_i$ for which $l_i = -1$, and $\mathcal{C}_1$ the vectors for which $l_i = +1$. The dataset is said to be linearly separable if there exists $\boldsymbol{\xi} \in \mathbb{R}^{2n'}$ and $\nu \in \mathbb{R}$ that define a hyperplane $\mathcal{P} \in \mathbb{R}^{2n'}$ that satisfies the following:

[1]The following derivation of the SVM classifier is generic, but notations are kept as similar as possible to the decoding problem, with the exception of l_i which denotes the binary label of the i-th element of the dataset and not the LLR as before.

$$\mathcal{P} : \{\tilde{\boldsymbol{y}} \in \mathbb{R}^{2n'} \mid f(\tilde{\boldsymbol{y}}) = \boldsymbol{\xi}^T \tilde{\boldsymbol{y}} + \nu = 0\} \tag{2.2}$$

$$\begin{aligned} \text{such that:} \quad & f(\tilde{\boldsymbol{y}}) < 0 \quad \text{for all } \tilde{\boldsymbol{y}} \in \mathcal{C}_0 \\ & f(\tilde{\boldsymbol{y}}) \geq 0 \quad \text{for all } \tilde{\boldsymbol{y}} \in \mathcal{C}_1. \end{aligned} \tag{2.3}$$

The SVM principle consists in producing a hyperplane $\mathcal{P}^*$ that satisfies the *maximum margin property*, i.e., being at an equal and maximum distance from the nearest points of each class (the so-called *support vectors*). Let $\tilde{\boldsymbol{y}}^*$ denote the closest point to the hyperplane $\mathcal{P}^*$ and belonging to any of the two classes. Without loss of generality, we can normalize $\boldsymbol{\xi}$ and ν so that:

$$|f(\tilde{\boldsymbol{y}}^*)| = |\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}^* + \nu| = 1. \tag{2.4}$$

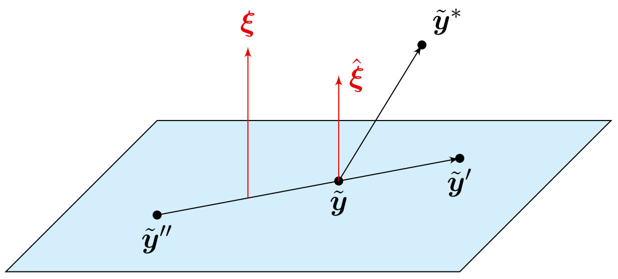

$\boldsymbol{\xi}$
$\tilde{\boldsymbol{y}}^*$
$\hat{\boldsymbol{\xi}}$
$\tilde{\boldsymbol{y}}'$
$\tilde{\boldsymbol{y}}$
$\tilde{\boldsymbol{y}}''$

Figure 2.2: Visual aid for the SVM deduction.

Let us compute the distance between $\tilde{\boldsymbol{y}}^*$ and the hyperplane (see Figure 2.2). Consider two points on the plane, $\tilde{\boldsymbol{y}}'$ and $\tilde{\boldsymbol{y}}''$. We have that:

$$\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}' + \nu = \boldsymbol{\xi}^T \tilde{\boldsymbol{y}}'' + \nu = 0 \;\Rightarrow\; \boldsymbol{\xi}^T (\tilde{\boldsymbol{y}}' - \tilde{\boldsymbol{y}}'') = 0, \tag{2.5}$$

and thus $\boldsymbol{\xi}$ is orthogonal to the plane $\mathcal{P}$. Consider a point $\tilde{\boldsymbol{y}}$ on the plane, and the normalized vector $\hat{\boldsymbol{\xi}} = \frac{\boldsymbol{\xi}}{||\boldsymbol{\xi}||}$. In this case, the distance between $\tilde{\boldsymbol{y}}^*$ and the plane can be computed as the projection of $\tilde{\boldsymbol{y}}^* - \tilde{\boldsymbol{y}}$ on $\hat{\boldsymbol{\xi}}$:

$$\begin{aligned} d(\tilde{\boldsymbol{y}}^*, \mathcal{P}) &= |\hat{\boldsymbol{\xi}}^T (\tilde{\boldsymbol{y}}^* - \tilde{\boldsymbol{y}})| && (2.6) \\ &= \frac{1}{||\boldsymbol{\xi}||} |\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}^* - \boldsymbol{\xi}^T \tilde{\boldsymbol{y}}| && (2.7) \\ &= \frac{1}{||\boldsymbol{\xi}||} |\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}^* + \nu - (\underbrace{\boldsymbol{\xi}^T \tilde{\boldsymbol{y}} + \nu}_{=0})| && (2.8) \\ &= \frac{1}{||\boldsymbol{\xi}||} |\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}^* + \nu| && (2.9) \\ &= \frac{1}{||\boldsymbol{\xi}||}, && (2.10) \end{aligned}$$

where we have used that $\tilde{\boldsymbol{y}} \in \mathcal{P}$ and $|\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}^* + \nu| = 1$. Hence, the optimization problem takes the following expression:

$$\begin{aligned} &\underset{\boldsymbol{\xi},\nu}{\operatorname{argmax}} \quad \frac{1}{||\boldsymbol{\xi}||}, \quad \boldsymbol{\xi} \in \mathbb{R}^{2n'}, \nu \in \mathbb{R} \\ &\text{subject to} \quad \min_{i\in[1:N]} |\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}_i + \nu| = 1, \end{aligned} \tag{2.11}$$

or, equivalently:

$$\begin{aligned} &\underset{\boldsymbol{\xi},\nu}{\operatorname{argmin}} \quad \frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{\xi}, \quad \boldsymbol{\xi} \in \mathbb{R}^{2n'} \\ &\text{subject to} \quad l_i(\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}_i + \nu) \geq 1, \quad \forall i \in [1:N], \end{aligned} \tag{2.12}$$

where we have used that $|\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}_i + \nu| = l_i(\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}_i + \nu)$. We then employ the Lagrangian formulation to include the constraint into the objective function:

$$\begin{aligned} &\underset{\boldsymbol{\xi},\nu,\boldsymbol{\alpha}}{\operatorname{argmax}} \quad \mathcal{L}(\boldsymbol{\xi},\nu,\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{\xi} - \sum_{i=1}^{N} \alpha_i (l_i(\boldsymbol{\xi}^T \tilde{\boldsymbol{y}}_i + \nu) - 1), \quad \boldsymbol{\xi},\boldsymbol{\alpha} \in \mathbb{R}^{2n'}, \nu \in \mathbb{R} \\ &\text{subject to} \quad \alpha_i \geq 0 \quad \forall i \in [1:N]. \end{aligned} \tag{2.13}$$

Next, we compute the gradient with respect to $\boldsymbol{\xi}$ and the derivative with respect to ν:

$$\begin{cases} \nabla_{\boldsymbol{\xi}} \mathcal{L} = \boldsymbol{\xi} - \sum_{i=1}^{N} \alpha_i l_i \tilde{\boldsymbol{y}}_i = 0 \;\Rightarrow\; \boldsymbol{\xi} = \sum_{i=1}^{N} \alpha_i l_i \tilde{\boldsymbol{y}}_i \\ \dfrac{\partial \mathcal{L}}{\partial \nu} = -\sum_{i=1}^{N} \alpha_i l_i = 0. \end{cases} \tag{2.14}$$

Finally, substituting these expressions into (2.13), we get a formulation of the optimization problem that only depends on $\boldsymbol{\alpha}$:

$$\begin{aligned} &\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \quad \mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} l_i l_j \alpha_i \alpha_j \tilde{\boldsymbol{y}}_i^T \tilde{\boldsymbol{y}}_j, \quad \boldsymbol{\alpha} \in \mathbb{R}^{2n'} \\ &\text{subject to} \quad \alpha_i \geq 0 \quad \forall i \in [1:N] \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i l_i = 0. \end{aligned} \tag{2.15}$$

The problem (2.15) is solved using quadratic programming, yielding a solution $\boldsymbol{\alpha}^\star = (\alpha_1, \alpha_2, ..., \alpha_N)$. This vector is then used to compute the hyperplane:

$$\boldsymbol{\xi}^\star = \sum_{i=1}^{N} \alpha_i^\star l_i \tilde{\boldsymbol{y}}_i, \tag{2.16}$$

and the Karush–Kuhn–Tucker (KKT) conditions are exploited to determine the value of ν:

$$\alpha_i^\star (l_i(\boldsymbol{\xi}^{\star T} \tilde{\boldsymbol{y}}_i + \nu^\star) - 1) = 0, \quad \forall\, i \in [1:N]. \tag{2.17}$$

$$\text{For any } j \in [1:N] \text{ such that } \alpha_j > 0 \;\Rightarrow\; l_i(\boldsymbol{\xi}^{\star T} \tilde{\boldsymbol{y}}_j + \nu^\star) = 1. \tag{2.18}$$

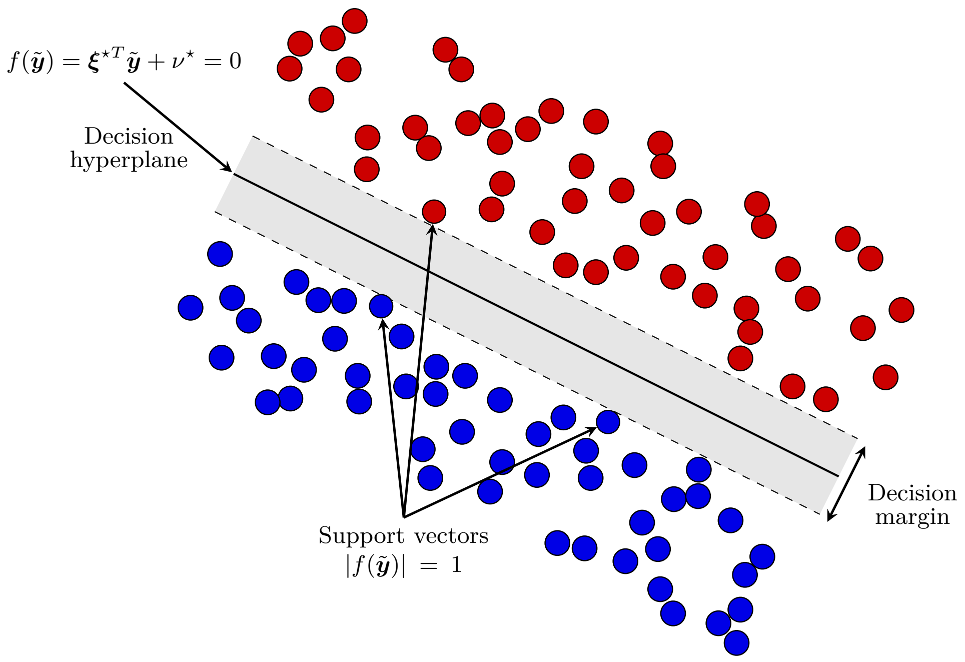

$f(\tilde{\boldsymbol{y}}) = \boldsymbol{\xi}^{\star T}\tilde{\boldsymbol{y}} + \nu^\star = 0$
Decision hyperplane
Decision margin
Support vectors $|f(\tilde{\boldsymbol{y}})| = 1$

Figure 2.3: Visual representation of an SVM classifier.

All the points that satisfy the condition (2.18) constitute the closest points to the hyperplane $\mathcal{P}$, and are called *support vectors*. We may solve (2.18) using any one of such points. Figure 2.3 shows a schematic representation of an SVM classifier. In sum, learning a classifier from the labeled dataset amounts to learning a *decision function* $f(\cdot)$ such that for all $\tilde{\boldsymbol{y}} \in \mathbb{R}^{2n'}$,

$$\begin{cases} f(\tilde{\boldsymbol{y}}) > 0 \implies \tilde{\boldsymbol{y}} \in \mathcal{C}_1 \\ f(\tilde{\boldsymbol{y}}) \leq 0 \implies \tilde{\boldsymbol{y}} \in \mathcal{C}_0. \end{cases} \tag{2.19}$$

2.1.3 Linearly non-separable data

In many applications, and depending on the distribution of the dataset, we may be interested in producing a non-linear classification function. In this scenario, we can resort to a technique called the *kernel method* or *kernel trick*: the basic idea consists in transforming the data into a high-dimensional space, where the data *becomes* linearly separable (see Figure 2.4). For instance, considering a dataset of vectors $\{\tilde{\boldsymbol{y}}_i\}_{i=1,\dots,N}$ of dimension 2, we could apply the following polynomial transformation:

$$\phi(\tilde{\boldsymbol{y}}) = \phi(\tilde{y}_1, \tilde{y}_2) = (1,\, \tilde{y}_1,\, \tilde{y}_2,\, \tilde{y}_1\tilde{y}_2,\, \tilde{y}_1^2,\, \tilde{y}_2^2), \tag{2.20}$$

which would allow for a more complex —and thus powerful— separating function. However, observe that in the final optimization problem (2.15), the objective function does not depend explicitly on the data $\tilde{\boldsymbol{y}}_i$, but rather on the pairwise Euclidean inner product $\langle \tilde{\boldsymbol{y}}_i, \tilde{\boldsymbol{y}}_j \rangle = \tilde{\boldsymbol{y}}_i^T \tilde{\boldsymbol{y}}_j$, for $i, j \in [1 : N]^2$. Following a similar mathematical deduction to that of Section 2.1.2, it is easy to conclude that, if we project the data onto a high-dimensional space through a non-linear function $\Phi(\cdot)$, the optimization problem (2.15) can be expressed as follows:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \quad & \mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} l_i l_j \alpha_i \alpha_j K(\tilde{\boldsymbol{y}}_i, \tilde{\boldsymbol{y}}_j), \quad \boldsymbol{\alpha} \in \mathbb{R}^{2n'} \\ \text{subject to} \quad & \alpha_i \geq 0 \quad \forall i \in [1:N] \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i l_i = 0, \end{aligned} \tag{2.21}$$

where $K(\tilde{\boldsymbol{y}}_i, \tilde{\boldsymbol{y}}_j) = \langle \Phi(\tilde{\boldsymbol{y}}_i), \Phi(\tilde{\boldsymbol{y}}_j) \rangle$ denotes the inner product in the image space defined by $\{\Phi(\tilde{\boldsymbol{y}}) : \tilde{\boldsymbol{y}} \in \mathbb{R}^{2n'}\}$. As a consequence of this, we do not need to explicitly transform the data into the high-dimensional space, but only apply pairwise inner products. In this work, we will employ the very well-known Radial Basis Function (RBF), defined as:

$$K(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{y}}') \triangleq e^{-\gamma ||\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{y}}'||^2}, \ \gamma \in \mathbb{R}^+, \tag{2.22}$$

which corresponds to the inner product over the infinite-dimensional space of exponentials.

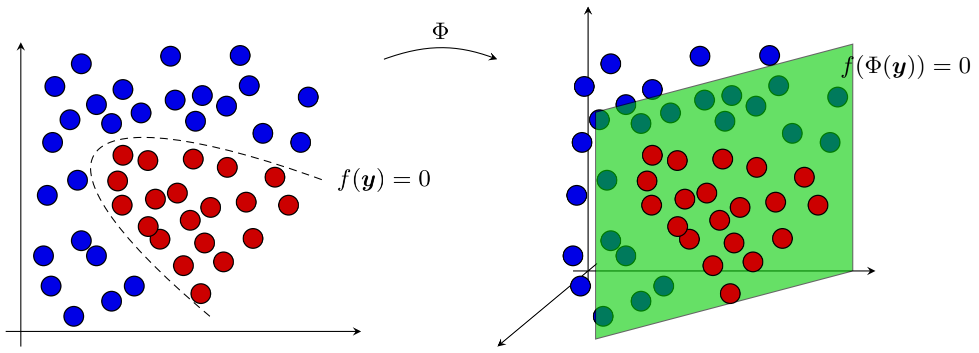


Figure 2.4: Visual representation of the kernel method. In its original form, the data is not linearly separable. However, through the transformation Φ, the space is now three-dimensional and the data can be separated with a plane.

Finally, let us observe that the absolute value $|f(\tilde{\boldsymbol{y}})|$ acts as a *reliability* measure: while $|f(\tilde{\boldsymbol{y}})| = 1$ indicates a support vector, $|f(\tilde{\boldsymbol{y}})| > 1$ indicates that the point is further away from the separating hyperplane, and thus is more likely to belong to that class.

2.2 Previous works on SVM-based decoding

Regardless of whether the dataset is linearly separable or not, SVMs are, by nature, binary classifiers. To build a decoding algorithm for an (n, k) linear block code, we need to resort to multiclass classification. Previous solutions [APP20; KB08; SY16] for SVM joint demodulation and decoding employ the so-called *one vs. rest* and *one vs. one* approaches [HL02]. We will now describe both approaches, along with their limitations and complexity constraints.

In what follows, we will assume that the training dataset $\{\tilde{\boldsymbol{y}}_i, l_i\}_{1 \leq i \leq N}$ contains several noisy realizations of every valid codeword, where:

- $\tilde{\boldsymbol{y}}_i \in \mathbb{R}^{2n'}$ denotes a received (originally complex) signal with its real and imaginary parts concatenated, corresponding to a transmitted message $\boldsymbol{u}_i$ as in Figure 2.1;
- l_i indicates the ground truth class to which $\tilde{\boldsymbol{y}}_i$ belongs, i.e., the transmitted message $\boldsymbol{u}_i$. A label l_i indicates belonging to the class $\mathcal{C}_i$, $i \in [1:2^k]$, which corresponds to a possible message (for instance, the conversion of the label l_i to binary frame of k bits).

2.2.1 One vs. rest

The *one vs. rest* method is based on producing 2^k binary SVM decision functions $f^{(j)}$ for $j \in [1:2^k]$, each one isolating one class (i.e., one codeword) against all the others [HL02]. This leaves a binary layout where the value $f^{(j)}(\tilde{\boldsymbol{y}})$ indicates whether the element $\tilde{\boldsymbol{y}}$ belongs to the class $\mathcal{C}_j$. We repeat this process for every class, and end up with 2^k SVM classifiers, each corresponding to a valid codeword. All the SVMs are then applied to the received signal $\tilde{\boldsymbol{y}}$, and the selected class $\mathcal{C}_{j^\star}$ (codeword) is such that:

$$j^\star = \underset{j\in[1:2^k]}{\operatorname{argmax}}\, f^{(j)}(\tilde{\boldsymbol{y}}). \tag{2.23}$$

This corresponds to the class that has the largest distance between the point $\tilde{\boldsymbol{y}}$ and the separating hyperplane. If no decision function $f^{(j)}(\tilde{\boldsymbol{y}})$ gives a positive outcome, then the nearest class is selected, i.e., $f^{(j)}(\tilde{\boldsymbol{y}})$ with the negative value closest to 0.

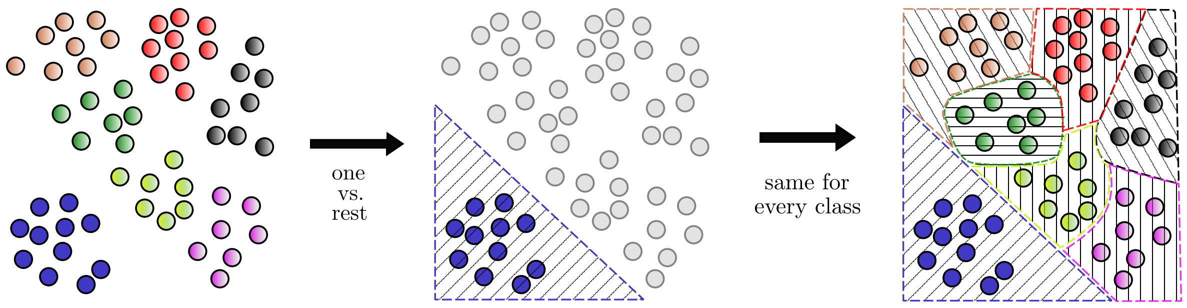


Figure 2.5: Visual representation of the one vs. rest approach. Each class is put against all others to produce a binary classifier. The procedure is repeated 2^k times, once for each valid codeword.

2.2.2 One vs. one

The one vs. one approach is employed in [KB08; Ram+09; SY16], and it consists in producing pairwise SVM classifiers for all possible combinations of valid codewords, yielding a set of functions $f^{(j,j')}\ \forall\, (j,j') \in [1:2^k]^2$. The total number of SVM functions can be easily computed:

$$\binom{2^k}{2} = \frac{2^k!}{2!(2^k-2)!} = \frac{2^k(2^k-1)\cancel{(2^k-2)!}}{2\cancel{(2^k-2)!}} = 2^{k-1}(2^k-1). \tag{2.24}$$

Once we have produced the $2^{k-1}(2^k-1)$ SVM binary classifiers, every function is applied to the received signal $\tilde{\boldsymbol{y}}$, and the selected class is determined through a voting system, given by:

$$C_{j^\star} = \underset{j\in[1:2^k]}{\operatorname{argmax}} \sum_{\substack{j'=1\\ j'\neq j}}^{2^k} \mathbb{1}\left\{f^{(j,j')}(\tilde{\boldsymbol{y}}) > 0\right\}. \tag{2.25}$$

Each decision function decides between two classes (codewords), and the class that gets the most votes at the end is selected. Ties can be resolved either randomly or by considering the mean of the votes' reliabilities, i.e., the distance to the separating hyperplanes.

2.3 Contributions: proposed system and training framework

In the following, we describe our suggested solution, which combines a bit-wise approach to the SVM decoder with a noiseless optimization procedure [DeB+24a]. Finally, a theoretical analysis is given that studies the relationship between our proposed solution and the bit-ML decoder.

2.3.1 Bit-wise SVM

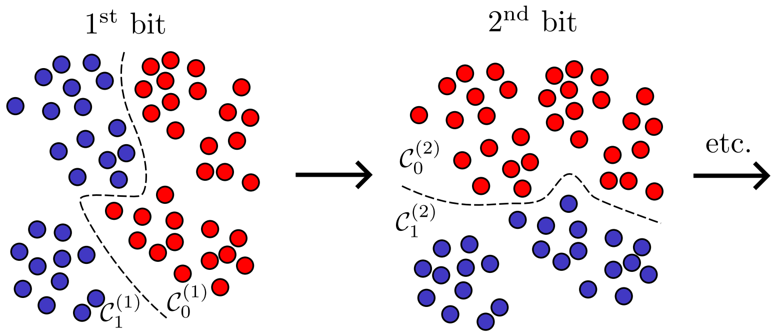


Figure 2.6: Visual representation of the proposed bit-wise approach.

The previous approaches present the main constraint of a complexity that increases exponentially with the message length, with at least 2^k SVMs for a code of size (n,k) —and even more for the one vs. one approach. To alleviate this constraint, we suggest a novel bit-wise approach that resorts to only k SVM classifiers for an (n,k) linear block code. This method transforms the multiclass problem generating one SVM per valid codeword, into a series of k binary classifications necessitating one SVM per bit position, as shown in Figure 2.6. To this end, for all $j \in [1:k]$, we divide the dataset $\{\tilde{\boldsymbol{y}}_1, ..., \tilde{\boldsymbol{y}}_N\}$ into two non-intersecting classes:

- $\mathcal{C}_1^{(j)}$ corresponding to the vectors for which the transmitted message $\boldsymbol{u} = (u_1, ..., u_k)$ satisfies $u_j = 1$;
- $\mathcal{C}_0^{(j)}$ corresponding to the vectors for which $u_j = 0$.

For each $j \in [1:k]$, an SVM classifier $f^{(j)}$ is produced such that if $f^{(j)}(\tilde{\boldsymbol{y}}) > 0$ then $\hat{u}_j = 1$, and $\hat{u}_j = 0$ otherwise. Consequently, each decision function $f^{(j)}$, for $j \in [1:k]$, will decide whether the jth bit of the estimated message $\hat{\boldsymbol{u}}$ is a 0 or a 1. Algorithm 1 outlines the decoding iterative process. The suggested bit-wise SVM not only reduces the number of SVMs necessary from 2^k to k, but can be implemented in parallel in order to reduce latency.

Algorithm 1 bit-wise SVM decoder

```
Input: $\boldsymbol{y}$ a noisy codeword of size $n$
Output: $\hat{\boldsymbol{u}}$ an estimated binary message of size $k$
Initialization: $f^{(j)}$ SVM bit classifier for $j \in [1:k]$
 1: $\tilde{\boldsymbol{y}} = [\mathcal{R}(\boldsymbol{y}), \mathcal{I}(\boldsymbol{y})]$
 2: $\hat{\boldsymbol{u}} = \vec{0}_k$
 3: for $j = [1:k]$ do
 4:     if $f^{(j)}(\tilde{\boldsymbol{y}}) \geq 0$ then
 5:         $\hat{u}_j = 1$
 6:     else
 7:         $\hat{u}_j = 0$
 8:     end if
 9: end for
10: return $\hat{\boldsymbol{u}}$
```

2.3.2 Proposed training: noiseless optimization

To further reduce the complexity with respect to the SVM-based solutions in [KB08; SY16] we make use of a particularly appealing feature of SVMs, namely, their maximum margin property, which yields separating hyperplanes that are equidistant from both dataset classes. As such, when investigating symmetric channel models like the AWGN, this is equivalent to a maximum margin classifier between *only* the original noiseless codewords (i.e. the classes' centroids). Consequently, rather than the traditional training approach which considers a dataset with randomly generated noisy codewords, it suffices to optimize —or *train*— the suggested bit-wise SVM on only noiseless modulated codewords as shown in Figure 2.7.

Observation 2.1. This noiseless training is possible due to the specific maximum margin property of SVM, and would not be possible in regular deep learning-based solutions, which would likely overfit to the noiseless codewords and perform poorly on noisy realizations.

The suggested noiseless optimization not only drastically reduces the size of the training dataset but also allows to be robust to possible mismatches between the training and actual channel conditions (e.g., different training and testing SNR). The new training dataset

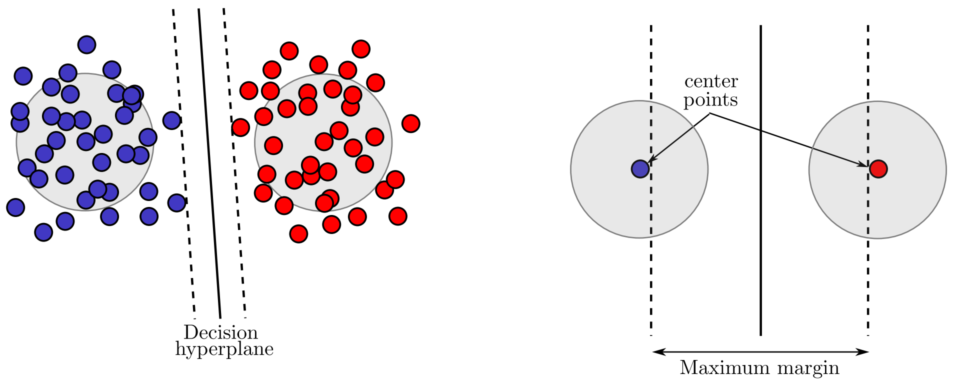


Figure 2.7: Visual demonstration of noiseless optimization: noisy scenario (left) and noiseless scenario (right).

contains only one sample per class, i.e., 2^k elements, which correspond to the noiseless modulated codewords $\{\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_{2^k}\}$, where the same processing of (2.1) is applied to every valid modulated codeword.

2.3.3 Proposed optimization problem

Combining the two suggested elements (bit-wise SVM and noiseless optimization), the training dataset will be composed of the 2^k valid modulated codewords $\{\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_{2^k}\}$ with $2n'$ elements each —where the same preprocessing (2.1) has been applied to $\boldsymbol{x}$—, and k binary classifiers will be produced following the bit-wise approach of Section 2.3.1. Because we are working with linearly non-separable data, the kernel method is employed, where the RBF is selected as the kernel function:

$$K(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{y}}') \triangleq e^{-\gamma||\tilde{\boldsymbol{y}}-\tilde{\boldsymbol{y}}'||^2},\ \gamma \in \mathbb{R}^+. \tag{2.26}$$

The classification of an unlabeled vector $\tilde{\boldsymbol{y}}$ consists of k classifiers $f^{(j)}(\tilde{\boldsymbol{y}})$, each determining the value of the jth bit of the estimated message $\hat{\boldsymbol{u}}$, and given by:

$$f^{(j)}(\tilde{\boldsymbol{y}}) = \sum_{i=1}^{2^k} l_i^{(j)} \alpha_i^{(j)} K(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}) + \nu^{(j)}, \tag{2.27}$$

where $l_i^{(j)} = +1$ if the jth information bit of the modulated codeword $\boldsymbol{x}_i$ is 1, and $l_i^{(j)} = -1$ otherwise. Lastly, $\boldsymbol{\alpha}^{(j)}$ constitutes the solution to the following optimization problem:

$$\begin{gathered}\underset{\boldsymbol{\alpha}}{\operatorname{argmin}}\ \frac{1}{2}\sum_{i,m=1}^{2^k} \alpha_i \alpha_m l_i^{(j)} l_m^{(j)} K(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_m) - \sum_{i=1}^{2^k} \alpha_i \\ \text{subject to: } \alpha_i \geqslant 0 \text{ and } \textstyle\sum_{i=1}^{2^k} l_i^{(j)} \alpha_i = 0.\end{gathered} \tag{2.28}$$

Each of the k bit-wise SVM optimization problems given in (2.28) can be written as a quadratic programming problem with linear constraints given by:

$$\begin{aligned} &\underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \ \frac{1}{2}\boldsymbol{\alpha}^T Q^{(j)}\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ &\text{subject to: } \boldsymbol{\alpha} \geq \mathbf{0} \text{ and } \boldsymbol{\alpha}^T \boldsymbol{l}^{(j)} = 0, \end{aligned} \tag{2.29}$$

where $Q^{(j)}$ is a matrix such that $Q^{(j)}_{i,m} = l_i^{(j)} l_m^{(j)} K(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_m)$. Since all $Q^{(j)}$ are definite positive matrices, these optimization problems are all convex.

2.3.4 Optimality analysis and interpretation

Although obtaining closed-form solutions of (2.28) for generic choices of the (n, k) linear block code, the constellation, and the parameter γ might be challenging, we show that under certain assumptions, the resulting SVM decision rule can be obtained in closed-form and related to the bit-ML decision rule.

Theorem 2.1 (Optimal solution and equivalence to bit-ML)**.**

(i) For $\gamma \gg 1$, the optimal solution to (2.28) *for all $j \in [1:k]$ is given by $\boldsymbol{\alpha}^* = (1, 1, ..., 1)$, and $\nu^* = 0$.*

(ii) Furthermore, if $\gamma = 1/\sigma^2$, this solution yields decision functions $f^{(j)}(\tilde{\boldsymbol{y}})$ equal to the bit-ML decision rule $g^{(j)}(\tilde{\boldsymbol{y}})$ of (1.36) *(equivalent to the bit-MAP with uniform and iid bits).*

Proof. In order to prove (i), let us notice that if $\gamma \gg 1$, then one can show that for all $i \neq m$, $K(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_m) \approx 0$. Hence, the objective function (2.29) can be expressed, for all $j \in [1:k]$, as

$$\frac{1}{2}\sum_{i=1}^{2^k} \alpha_i^2 - \sum_{i=1}^{2^k} \alpha_i = \frac{1}{2}\sum_{i=1}^{2^k} (\alpha_i^2 - 1)^2 - 2^{k-1}. \tag{2.30}$$

One can then easily show that the solution $\boldsymbol{\alpha}^* = (1, 1, ..., 1)$ yields a lower bound to the objective function, since $\sum_{i=1}^{2^k} (\alpha_i^2 - 1)^2 \geq 0$, and verifies the inequality constraints $\alpha_i \geq 0 \ \forall i \in [1:2^k]$. The equality constraint is also verified, since for all $j \in [1:k]$, each of the two classes $\mathcal{C}_1^{(j)}$ and $\mathcal{C}_0^{(j)}$ consist in 2^{k-1} sequences $\tilde{\boldsymbol{x}}_i$ and hence,

$$\sum_{i=1}^{2^k} l_i^{(j)} = |\mathcal{C}_0^{(j)}| - |\mathcal{C}_1^{(j)}| = 0 \quad \forall j \in [1:k]. \tag{2.31}$$

Thus, for all $j \in [1:k]$, $\boldsymbol{\alpha}^* = (1, 1, ..., 1)$ is the solution to the optimization problem in (2.28). As for $\nu^{(j)}$, note that by evaluating (2.27) for any $\tilde{\boldsymbol{x}}_m$ using $\gamma \gg 1$, we obtain

$$f^{(j)}(\tilde{\boldsymbol{x}}_m) = \sum_{i=1}^{2^k} l_i^{(j)} e^{-\gamma ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_m||^2} + \nu^{(j)} = l_m^{(j)} + \nu^{(j)}, \tag{2.32}$$

which yields $\nu^{(j)} = 0$.

To prove (ii), let us observe that replacing the proposed solution $\{\boldsymbol{\alpha}^* = (1, 1, ..., 1), \nu^* = 0\}$ in (2.27) yields

$$f^{(j)}(\tilde{\boldsymbol{y}}) = \sum_{i=1}^{2^k} l_i^{(j)} e^{-\gamma ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}||^2}, \tag{2.33}$$

where $l_i^{(j)}$ denotes the label of the point $\tilde{\boldsymbol{x}}_i$ for the jth classification problem (related to the value of its jth bit). Let us divide the summation argument into the two classes $\mathcal{C}_1^{(j)}$ and $\mathcal{C}_0^{(j)}$. Hence, we can rewrite the decision function as

$$f^{(j)}(\tilde{\boldsymbol{y}}) = \sum_{\tilde{\boldsymbol{x}}_i \in \mathcal{C}_1^{(j)}} e^{-\gamma ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}||^2} - \sum_{\tilde{\boldsymbol{x}}_i \in \mathcal{C}_0^{(j)}} e^{-\gamma ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}||^2}. \tag{2.34}$$

From (2.34), it is easy to deduce the value of the jth bit as

$$g^{(j)}(\tilde{\boldsymbol{y}}) = \mathbb{1}\left\{ \sum_{\tilde{\boldsymbol{x}}_i \in \mathcal{C}_1^{(j)}} e^{-\gamma ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}||^2} > \sum_{\tilde{\boldsymbol{x}}_i \in \mathcal{C}_0^{(j)}} e^{-\gamma ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}||^2} \right\}. \tag{2.35}$$

Selecting $\gamma = 1/\sigma^2$ in (2.35) yields the bit-ML rule in (1.36) for an AWGN channel with noise power σ^2 by noticing that

$$P_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}|\boldsymbol{x}(\boldsymbol{u})) = \frac{1}{(\pi\sigma^2)^{n'}} e^{-\dfrac{||\boldsymbol{x}(\boldsymbol{u}) - \boldsymbol{y}||^2}{\sigma^2}}. \tag{2.36}$$

$\square$

In the following, we will show that the assumption of $\gamma \gg 1$ in Theorem 2.1, is valid even for moderate SNR values. Besides, we will analyze the effect of relaxing the constraint $\gamma = 1/\sigma^2$ on the obtained results with respect to the bit-ML.

2.4 Numerical results and analysis

This section will present the simulation results obtained using the proposed approach, combining the bit-wise decoding framework and the noiseless optimization. The system model of Section 2.1.1 was employed along with a Monte Carlo simulation to compute the Bit Error Rate (BER), with a stopping criterion of 1000 frame errors for each considered E_b/N_0. Optimization problems are solved using the CVX modeling system for convex optimization [GB08; GB14]. The parity-check matrices were taken from the channel code database in [Hel+19].

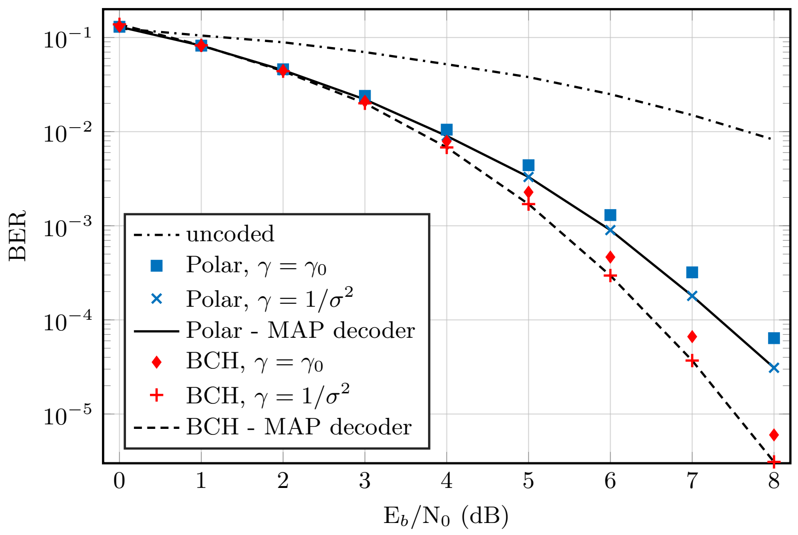


Figure 2.8: BER of the suggested decoder for both a BCH and a polar code of size $(32, 11)$, with a 16-QAM modulation and under an AWGN channel.

2.4.1 BER and effect of the hyperparameter γ

The suggested bit-wise SVM was implemented for both an extended $(32, 11)$ BCH code and a $(32, 11)$ Polar code, each under a 16-QAM modulation scheme and an AWGN channel with an $E_b/N_0 = \frac{n}{k.m}\frac{1}{\sigma^2}$, where $m = 4$ denotes the order of the modulation[2]. Let us define γ_s the value of the exponential's slope:

$$\gamma_s = 1/\sigma_s^2, \tag{2.37}$$

where σ_s^2 is the noise power such that $E_b/N_0 = s\,\text{dB}$. This is what is referred to as a value of γ *adapted* to a normalized signal-to-noise ratio of $E_b/N_0 = s\,\text{dB}$. In the following, we will distinguish two training scenarios. In the first scenario, the choice of γ in the RBF kernel is adapted to each E_b/N_0, i.e., $\gamma = 1/\sigma^2$, where σ^2 is the noise power corresponding to each E_b/N_0 ratio. In the second scenario, s is set to 0, i.e., $\gamma = \gamma_0$, for all values of E_b/N_0. The BER curves of both the suggested solution (bit-wise SVM) and the optimal solution (bit-ML) are given in Figure 2.8.

We observe that, for the first scenario, by adapting the value of γ to the corresponding E_b/N_0, the SVM decoder is matched to the bit-ML decision rule, and so their performances coincide as per Theorem 2.1. However, for the second scenario in which $\gamma = \gamma_0$, the resulting SVM curve degrades in the high E_b/N_0 (low-noise) regime. This is because the RBF kernel is fixed with $\gamma = \gamma_0$, and does not perfectly adapt to higher values of E_b/N_0.

To assess the effect of the choice of the parameter γ in the second scenario, Figure 2.9 shows the E_b/N_0 corresponding to a BER of 10^{-3} as a function of $s \in [-2 : 15]$ for both the

[2]Observe that the *one vs. rest* and *one vs. one* approaches described in Section 2.2 were not implemented due to their very high complexity, even for the code size considered. For results on shorter codes, the reader is referred to [KB08; SY16].

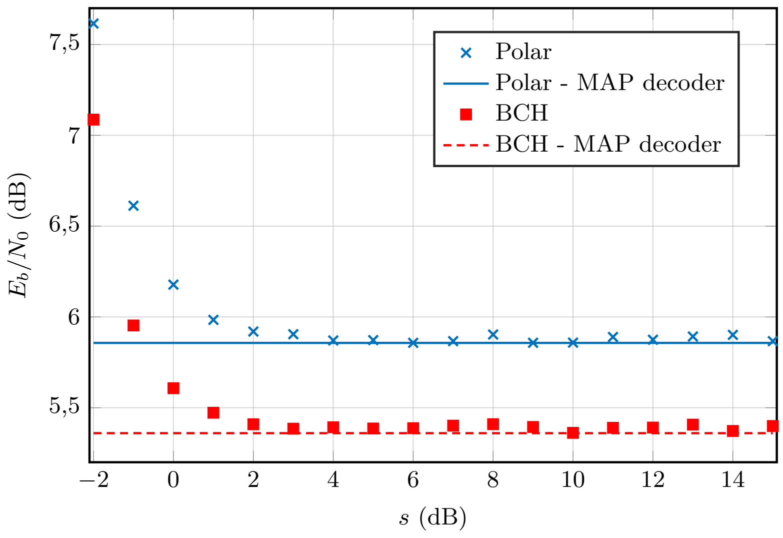


Figure 2.9: E_b/N_0 corresponding to a BER of 10^{-3} as a function of s.

$(32, 11)$ BCH code and the $(32, 11)$ polar code. One can observe that very low values of s —and thus their corresponding values of γ_s—, display poorer performances in terms of BER. However, above a threshold value of around 2dB, the SVM achieves the objective BER of 10^{-3} at essentially the same E_b/N_0 as the ML decoding solution. This phenomenon suggests that training with large values of γ relaxes the need to adapt γ to the current E_b/N_0, generalizing in this way the result of Theorem 2.1, (ii).

Moreover, as previously discussed, the result of Theorem 2.1, (i) is valid for γ corresponding to even moderate values of E_b/N_0. Figure 2.10 shows the solution to the optimization problem (2.28) as a function of $s \in [-2 : 15]$. We observe that, indeed, the optimal solution to (2.28) is given by $\boldsymbol{\alpha}^* = (1, 1, ..., 1)$ and $\nu^* = 0$ for all $s >$ 2dB, which corresponds to relatively moderate values of E_b/N_0.

2.4.2 Complexity analysis

Table 2.1 summarizes the decoding complexity of our method and those in the literature. As we can observe, the bit-wise approach is the first to enable a linearly growing number of SVMs, which is more easily scalable than exponentially growing methods. The same goes for the dataset: for the SVM to learn a decision rule between two classes, it has to *see* at least one element of each class. With our noiseless optimization, the dataset size has been reduced to its minimum $N = 2^k$.

Nevertheless, complexity is not only based on the number of SVM classifiers but also on the number of operations required to perform each one of these classifications. Even with our method with reduced complexity, the size of the dataset is 2^k, with one element per valid codeword. This implies exponential growth, as the size of the dataset determines the number of terms in (2.27).

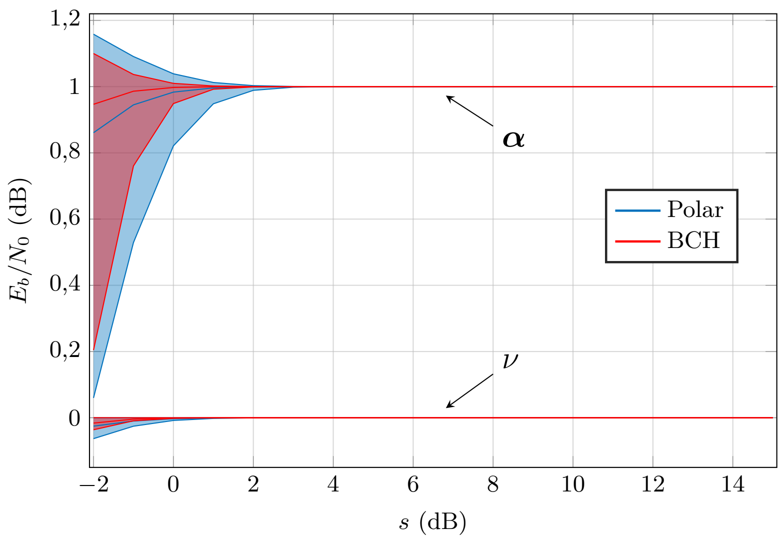


Figure 2.10: Optimal values of $\boldsymbol{\alpha}$ and ν —i.e. solutions to the optimization problem (2.28)— as a function of s.

	Bit-wise	One vs. rest	One vs. one
# of SVM classifiers	k	2^k	$2^{k-1}(2^k-1)$
# of terms in (2.27)	N	N	$\approx \frac{N}{2^{k-1}}$
# of terms in (2.27) with noiseless opt.	2^k	2^k	2

Table 2.1: Complexity comparison between methods.

2.5 Conclusion and perspectives

In this Chapter, we have discussed the application of SVMs to the problem of channel decoding. After providing the reader with an introduction to SVMs and a state-of-the-art study on their applications to channel decoding, we have presented our proposed solution and training framework. This novel decoding system improves performance and reduces complexity compared to the previous SVM-based methods:

(i) The number of decision functions is reduced from (at least) 2^k to k.

(ii) The number of elements in the dataset is reduced to a minimum of one element per possible codeword, i.e., 2^k.

(iii) In symmetric channels, robustness to the dataset's distribution is improved by only training on the noiseless codewords.

However, for the AWGN channel, our system is proven to be exactly equivalent to the ML decoder, which implies an exponentially growing complexity. This closes a door on the *SVM decoder for AWGN* debate, but also raises several questions:

(i) Could there be a way of training the SVM on a subset of the code instead of on the entire codeword space?

(ii) Is there a channel model (e.g., frequency or time selective, fading, unknown, etc.) where the ML decoding rule is unavailable in closed-form, but where the SVM is able to find an optimal or pseudo-optimal decoder in a data-driven manner?

(iii) Could we make a small sacrifice in performance (e.g., using a different kernel method and/or dataset) to reduce the number of support vectors and, thus, the overall complexity of the decoder?

These and many other questions may be addressed in future works, potentially combining other machine learning-based techniques to, for instance, learn smarter feature representations for the codeword space that reduce the complexity of the final SVM decision function.

Chapter 3

Syndrome-based neural decoding for BPSK

Contents

Recall that one of the primary challenges hindering the application of machine learning-based solutions to channel decoding is *scalability*, referring to the capacity of a neural network to learn an effective decoding rule within the boundaries of limited training, computing and storage capacity. One aspect of this, regarding the multiplicity of noise realizations needed for learning, was discussed in Chapter 2 and an SVM-based decoder was proposed that is trained over noiseless codewords. The other two key features that configure the scalability problem pertain to the codeword space —which grows exponentially with the code dimension— and the network's size —which tends to increase dramatically to be able to adapt to larger codes. These are the aspects that are addressed in the current Chapter.

Regarding the codeword space, a successful decoding technique is the so-called *model-free* approach, proposed by Bennatan et al. in [BCK18a], which makes use of a multiplicative noise model previously depicted in [RU01]. This approach, which relies on the symmetry of the BPSK modulation scheme, renders the training procedure independent from the transmitted codeword. This enables the *single-codeword training property*, which completely removes the need to train over all valid codewords by relying uniquely on one codeword throughout the entire training process. The authors in [BCK18a] employ Multi-Layer Perceptrons (MLP) and Recurrent Neural Networks (RNN) for the neural estimator, and display very promising performances for short and medium-length codes (up to 127 bits). Subsequently, the model-free approach was adopted by several authors: a study in 2022 by Choukroun et al. introduced a transformer-based architecture [CW22a], therein referred to as Error Correction Code Transformer (ECCT), that reduces the number of parameters and improves decoding performances in some cases. The ECCT was revisited by Park et al. in 2023 where two distinct parity-check matrices are used to diversify the information extracted by the neural network [Par+23]. Other works have also explored iterative approaches, e.g., employing the original decoder several times in a sequential manner [KP20], or applying denoising diffusion models [CW23]. However, these decoders present shortcomings in their construction: (i) they work on a codeword level, assigning the same importance to information bits and parity bits during the estimation; (ii) because of this, when applied to non-systematic codes (e.g. Polar codes in their original form), they require further calculations to obtain the message; (iii) the parity-check matrix construction is not discussed, despite the fact that its structure has significant impact on the decoder's training and performance. These elements constitute potential sources of inefficiency or suboptimal performance that will be addressed in this Chapter.

With respect to the network's size, the original work in [BCK18a] uses an RNN as the high-performance solution but its size increases exponentially with the code length, employing up to 20 million weights for a BCH code of size $(127, 64)$. The transformer-based architecture addresses this by proposing a solution with a more constant number of weights, employing around 2 million weights for the same BCH code (and around 1.5 million for shorter codes). In this Chapter, we also attempt to further reduce the network's complexity to take another step toward realistic implementations, whilst minimizing performance penalties.

In light of the two elements previously described, the contributions presented in this Chapter can be classified as follows:

(i) *Message-oriented decoder*: building from the model-free approach, we introduce a novel decoding framework that displays significant improvements by considering only the information bits instead of the whole codeword. We define a pseudo-inverse operation on the codewords that allows the network to directly estimate the errors in the message, thus improving decoding performances and rendering the decoder directly applicable to non-systematic codes.

(ii) *Parity-check matrix*: we employ information theory-based metrics to assess the influence of the parity-check matrix on the decoder's training and performance, and propose a simple algorithm to build a more suitable parity-check matrix according to these metrics.

(iii) *Recurrent ECCT*: with respect to the network's size, we introduce a recurrent version of the transformer-based architecture proposed in [CW22a], which reduces the total number of parameters in the NN to only a fraction of the ones used in previous architectures, while maintaining competitive decoding performances.

The work in this Chapter is organized as follows. We start by offering an overview of the system layout and the model-free (also referred to as *syndrome-based*) decoding approach. For this purpose, we start by presenting the multiplicative noise model of [RU01, Lemma 1], followed by the model-free approach introduced by Bennatan et al. in [BCK18a]. This brief state-of-the-art study is completed with an overview of the main deep-learning architectures found in the literature. Subsequently, we present the main limitations associated with the previous decoder and propose a novel *message-oriented* decoding framework that displays significant improvements, especially when employing the low-complexity architectures. We continue by proposing the new recurrent transformer-based network, followed by a complexity analysis carried out among the considered architectures regarding the total number of parameters that configure each network. Finally, we study the influence of the parity-check matrix on the decoder and propose an algorithm to modify it accordingly. All of these contributions are thoroughly evaluated —separately and jointly— by applying them to Polar and BCH codes of different sizes and code rates.

Most of the work carried out in this Chapter can be found in [DB23] and [DeB+24d].

3.1 Model-free decoding approach

As stated in the Introduction, the purpose of this work is to employ machine learning-based solutions to produce channel decoders that offer competitive performances while maintaining relatively low complexity and latency. This Section will describe the model-free technique introduced in 2018 by Bennatan et al. [BCK18a], along with the necessary tools to implement it under a BPSK modulation scheme.

3.1.1 System model

Let us start by briefly introducing the framework. A simplified schematic is given in Figure 3.1. Let $\boldsymbol{u} \in \{0,1\}^k$ denote the k-bit message to be transmitted, and $\boldsymbol{c} \in \{0,1\}^n$ its associated n-bit codeword through a linear FEC code $\mathcal{C}$. This codeword is mapped into a BPSK vector $\boldsymbol{x} = 1 - 2\boldsymbol{c}$, which is transmitted through a symmetric binary-input AWGN channel. The received signal $\boldsymbol{y}$ is used as input to the decoder to give an estimate $\hat{\boldsymbol{u}}$ of the message $\boldsymbol{u}$.

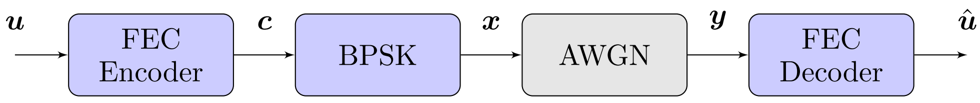


Figure 3.1: General system model for model-free BPSK decoding.

Notation. In the following, for $x \in \mathbb{R}$, x^s denotes its sign (i.e., $x^s = +1$ if $x > 0$, and $x^s = -1$ otherwise), and x^b represents its binary hard decision (i.e., $x^b = 0$ if $x > 0$, and $x^b = 1$ otherwise). The same reasoning is applied element-wise for a vector $\boldsymbol{x} \in \mathbb{R}^n$.

3.1.2 Noise model

Let us now present the multiplicative noise model from [RU01, Lemma 1]. In the traditional AWGN channel model, the received random signal is expressed as follows:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{w}, \tag{3.1}$$

where $\boldsymbol{x}$ is a random vector of size n that represents the BPSK modulated codeword and $\boldsymbol{w} = (w_1, w_2, ..., w_n)$, such that $\{w_i\}_{1\leq i\leq n}$ are independent and identically distributed (iid) random variables distributed as $\mathcal{N}(0, \sigma^2)$. In this scenario, the following holds for all $i \in [1 : n]$:

$$\begin{align} P(Y_i^s \neq X_i) &= P(Y_i^s = 1|X_i = -1)P(X_i = -1) + P(Y_i^s = -1|X_i = 1)P(X_i = 1) \tag{3.2}\\ &= P(W_i > 1)\frac{1}{2} + P(W_i < -1)\frac{1}{2} \tag{3.3}\\ &= P(W_i > 1) = P(W_i < -1). \tag{3.4} \end{align}$$

In this work, in order to motivate the preprocessing of the decoder input $\boldsymbol{y}$, we need to introduce an equivalent multiplicative formulation of the AWGN channel, which we define following [RU01, Lemma 1] by

$$\boldsymbol{y} = \boldsymbol{x} \odot \boldsymbol{z}, \tag{3.5}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ designate the channel input and output vectors, $\odot$ denotes the Hadamard (element-wise) product, and $\boldsymbol{z}$ is a random noise that verifies, $\forall i \in [1 : n]$:

$$\begin{align} P_{Z_i}(z_i) &= P_{Y_iX_i}(z_i) \tag{3.6}\\ &= \sum_{x=-1,1} P_{X_i}(x)P_{Y_iX_i|X_i}(z_i|x) \tag{3.7}\\ &= \frac{1}{2}P_{Y_i|X_i}(z_i|1) + \frac{1}{2}P_{-Y_i|X_i}(z_i|-1) \tag{3.8}\\ &= \frac{1}{2}P_{Y_i|X_i}(z_i|1) + \frac{1}{2}P_{Y_i|X_i}(-z_i|-1) \tag{3.9}\\ &= \frac{1}{2}P_{Y_i|X_i}(z_i|1) + \frac{1}{2}P_{Y_i|X_i}(z_i|1) \tag{3.10}\\ &= P_{Y_i|X_i}(z_i|1) \tag{3.11}\\ &= P_{Y_i|X_i}(y_i|1). \tag{3.12} \end{align}$$

Therefore, $Z_i \sim \mathcal{N}(1, \sigma^2)$ for all $i \in [1 : n]$. As a direct consequence of this multiplicative model for the noise, the probability of error simplifies to:

$$P(Y_i^s \neq X_i) = P(Z_i < 0). \tag{3.13}$$

It is easy to observe that the probability of a 0-centered white Gaussian noise being greater than 1 is the same as a 1-centered white Gaussian noise of the same power being smaller than 0, i.e., (3.4) and (3.13) are equal for a same given variance σ^2.

3.1.3 Decoding framework

In [BCK18a], the authors proposed to use the noise model from Section 3.1.2 and the decoding framework depicted in Figure 3.2. Instead of using the channel output directly to estimate the transmitted codeword —and thus encountering the dimensionality problem—, we use this decoding approach that transforms the channel output $\boldsymbol{y}$ into two vectors: the absolute value $|\boldsymbol{y}|$ and the syndrome $H\boldsymbol{y}^b$, where H denotes a parity check matrix for the code $\mathcal{C}$.

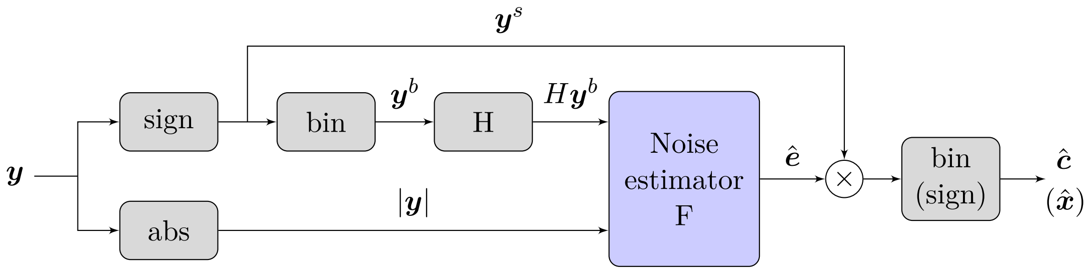


Figure 3.2: Decoder framework proposed in [BCK18a].

Using this approach, a noise estimator (that will later be implemented using deep neural networks) outputs a vector $\hat{\boldsymbol{e}}$, which estimates the so-called *bitflip* vector[1]. This vector indicates the positions of the received signal $\boldsymbol{y}$ that have suffered bitflips, i.e., the indices where $y^b \neq c$, or equivalently, $z < 0$. In [BCK18a], the authors used negative values for positions where a bitflip occurred (so that the product with $\boldsymbol{y}^s$ would correct them), and positive values elsewhere. For our contribution, we will choose a slightly different representation. The final estimate of the transmitted codeword $\boldsymbol{c}$ is given by:

$$\hat{\boldsymbol{c}} = \text{bin}(\boldsymbol{y}^s \hat{\boldsymbol{e}}), \tag{3.14}$$

or equivalently, the estimate of the transmitted BPSK signal $\boldsymbol{x}$ is given by:

$$\hat{\boldsymbol{x}} = \text{sign}(\boldsymbol{y}^s \hat{\boldsymbol{e}}). \tag{3.15}$$

3.1.4 Optimality analysis

The main result from Bennatan et al. is proving that the decoding framework from Section 3.1.3 does not incur any loss of optimality, i.e., the inputs $|\boldsymbol{y}|$ and $H\boldsymbol{y}^b$ are *sufficients statistics* for optimal decoding. This is outlined in the following theorem.

Theorem 3.1 (Sufficient statistics)**.** *Consider the decoding framework from Section 3.1.3, where the transmitted codeword* $\mathbf{c}$ *is estimated (or equivalently, the BPSK signal* $\boldsymbol{x}$*). In this scenario, the following equation holds, for all* $i \in [1:n]$*:*

$$P_{X_i|\boldsymbol{Y}}(x|\boldsymbol{y}) = P_{Z_i^s|\,|\boldsymbol{Z}|,H\boldsymbol{Z}^b}(xy_i^s|\,|\boldsymbol{y}|, H\boldsymbol{y}^b), \tag{3.16}$$

[1]To enhance clarity, we have opted for a different notation than the one employed in the original work by Bennatan et al.

where $Z_i \sim \mathcal{N}(1, \sigma^2)\ \forall\, i \in [1:n]$ *is defined as in Section 3.1.2.*

Proof. The proof can be found in [BCK18b]. □

The result in (3.16) provides independency from the transmitted symbol sequence $\boldsymbol{x}$, as the pdf expression given by $P_{Z_i^s \,|\, |\boldsymbol{Z}|, H\boldsymbol{Z}^b}$ only depends on the noise vector $\boldsymbol{Z}$. As a result, the decoding system can be trained on noisy realizations of any given codeword (for instance, the all-zero codeword).

3.1.5 Neural network architectures for the noise estimator

As seen in Section 3.1.3, the noise estimator will take two inputs —$|\boldsymbol{y}|$ and $H\boldsymbol{y}^b$— and give an estimate $\hat{\boldsymbol{e}}$ of the bitflip positions. This section presents a few Deep Neural Network (DNN) architectures implemented in the literature. The input to every DNN is a vector consisting of the concatenation of the two considered variables from Theorem 3.2, i.e., $(\,|\boldsymbol{y}|, H\boldsymbol{y}^b\,)$.

Observation 3.1. The authors in [BCK18a] call the estimator *Noise Estimation* and the output is denoted $\hat{\boldsymbol{z}}$, as if it was an estimate of the noise $\boldsymbol{z}$. This is not the case, as the network does not seek to estimate the actual value of the noise but rather only the positions where a bitflip occurred. This is why we have defined an extra notation vector $\boldsymbol{e}$ called the bitflip vector, and the network outputs its estimate $\hat{\boldsymbol{e}}$, which should indicate the positions where $z < 0$.

3.1.5.1 Multi-layer Perceptron

The MLP is the classical feed-forward and dense DNN, and was Bennatan's first implemented architecture (where it is called *Vanilla Multi-Layer*), with the only difference that the input is fed to each layer, in addition to the output of the previous layer, in order to mimic the behavior of the BP algorithm. The network consists of d_l fully connected layers with a Rectified Linear Unit (ReLU) activation function. Each hidden layer contains αn neurons, where (n, k) are the code parameters and $\alpha \in \mathbb{N}^+$ is a scaling factor to be selected. A final output layer contains n neurons, and the hyperbolic tangent activation function is employed to output values in the range $[-1, +1]$. The architecture is displayed in Figure 3.3.

3.1.5.2 Recurrent Neural Networks

The authors in [BCK18a] also proposed an RNN architecture as a more memory-efficient alternative. The network consists in stacking recurrent cells on top of each other. The number of cells to be stacked will define the depth of the neural network. Observe that RNNs are usually employed for sequential data. In our case, the input vector is fed repeatedly throughout all the time steps.

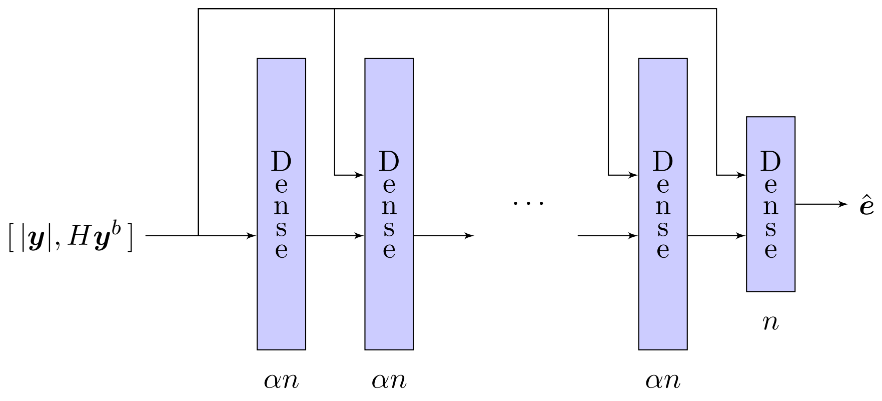


Figure 3.3: MLP-based architecture for the noise estimator. The number of neurons (i.e., the output size) is indicated below each layer, where $\alpha \in \mathbb{N}^+$, and n denotes the code's block length.

The basic architecture is depicted in Figure 3.4, where d_l recurrent layers are stacked on top of each other and perform T time steps before producing an output $\hat{\boldsymbol{e}}$. Each cell $\boldsymbol{g}_i$, $\forall i \in [1 : d_l]$ is composed of several Gated Recurrent Units (GRU) [Cho+14], which is a more lightweight variant to Long Short-Term Memory (LSTM) cells [HS97] due to their lack of a reset gate. $\boldsymbol{h}_{i,t}$ designates the state vector of the ith GRU cell at time t. Each GRU cell is again composed of αn GRU units, and the final dense layer has n neurons and employs also a hyperbolic tangent activation function. Differently from [BCK18a], a final dense layer with n units is added at the end of the RNN to obtain an output with the correct size[2].

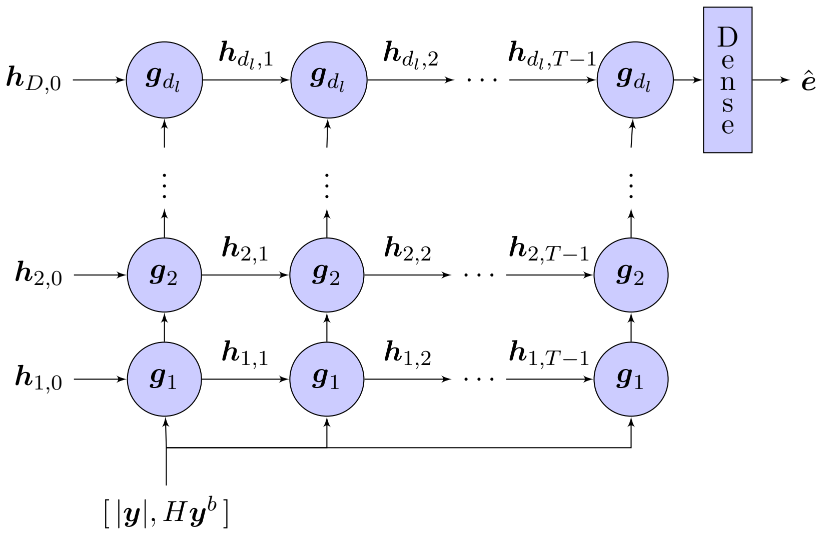
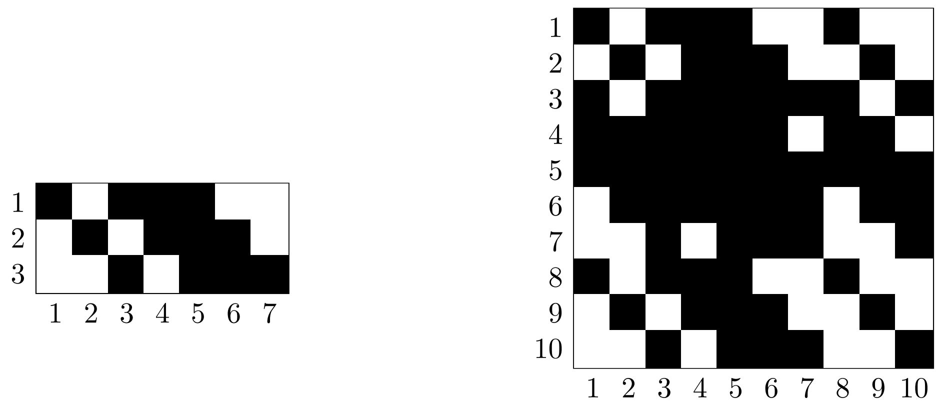


Figure 3.4: RNN-based architecture for the noise estimator. The architecture is slightly changed with respect to [BCK18a], adding a final dense layer to obtain the correct output dimension.

[2]It is not clear how the authors of [BCK18a] obtain an output of length n without this dense layer. In any case, the performances obtained match the expected ones.

3.1.5.3 Transformer

In a subsequent study, Choukroun et al. proposed a novel transformer-based architecture for the bitflip estimator [CW22a] —therein referred to as ECCT— inspired by the work of Vaswani et al. [Vas+17] and depicted in Figure 3.6. This network consists of three main stages: (i) an embedding stage, where each component of the input vector is projected into a high-dimensional space of size d_e; (ii) an encoding stage, repeated N times, where a masked Multi-Head Self-Attention (MH-SA) block alternates with two dense Feed-Forward Neural Networks (FFNN) —an expanding and a contracting one— with normalization layers added before according to Figure 3.6; and (iii) a decoder stage, which consists in a normalization layer, a dense layer to squeeze the embedding dimension to 1 and a final dense layer that outputs the estimate $\hat{\boldsymbol{e}}$. The mask is a symmetric binary matrix of size $(2n-k) \times (2n-k)$ that contains a one in the position (i, j) if the ith and jth elements of the input vector $[|\boldsymbol{y}|, H\boldsymbol{y}^b]$ are *connected*. To achieve this, we start with an identity matrix I_{2n-k}, since every element is connected to itself. Then, for each row $i \in [1:n-k]$ of the parity-check matrix, for every pair $(j, j') \in [1:n] \times [1:n]$ such that $\mathrm{H}_{i,j} = \mathrm{H}_{i,j'} = 1$, we unmask the positions (j, j') and $(n+i, j)$, along with their symmetric elements. The resulting matrix can be seen as an extended adjacency matrix of the Tanner graph, in order to account for all the $2n-k$ input elements. For further details, the reader is referred to [CW22a].



(a) Parity-check matrix. (b) Mask.

Figure 3.5: Example of a parity-check matrix of a Hamming code of size $(7, 4)$ and its corresponding mask for the MH-SA block. Black squares correspond to ones and white squares to zeros.

3.2 Contributions to the model-free decoder

As previously stated, one shortcoming of the decoder from Section 3.1.3 is that the entire codeword $\boldsymbol{c}$ is estimated, and thus the same importance is given to both information and parity bits. Besides, when employing non-systematic codes, an additional series of linear operations has to be applied to the codeword, which may cause a performance penalty. Finally,

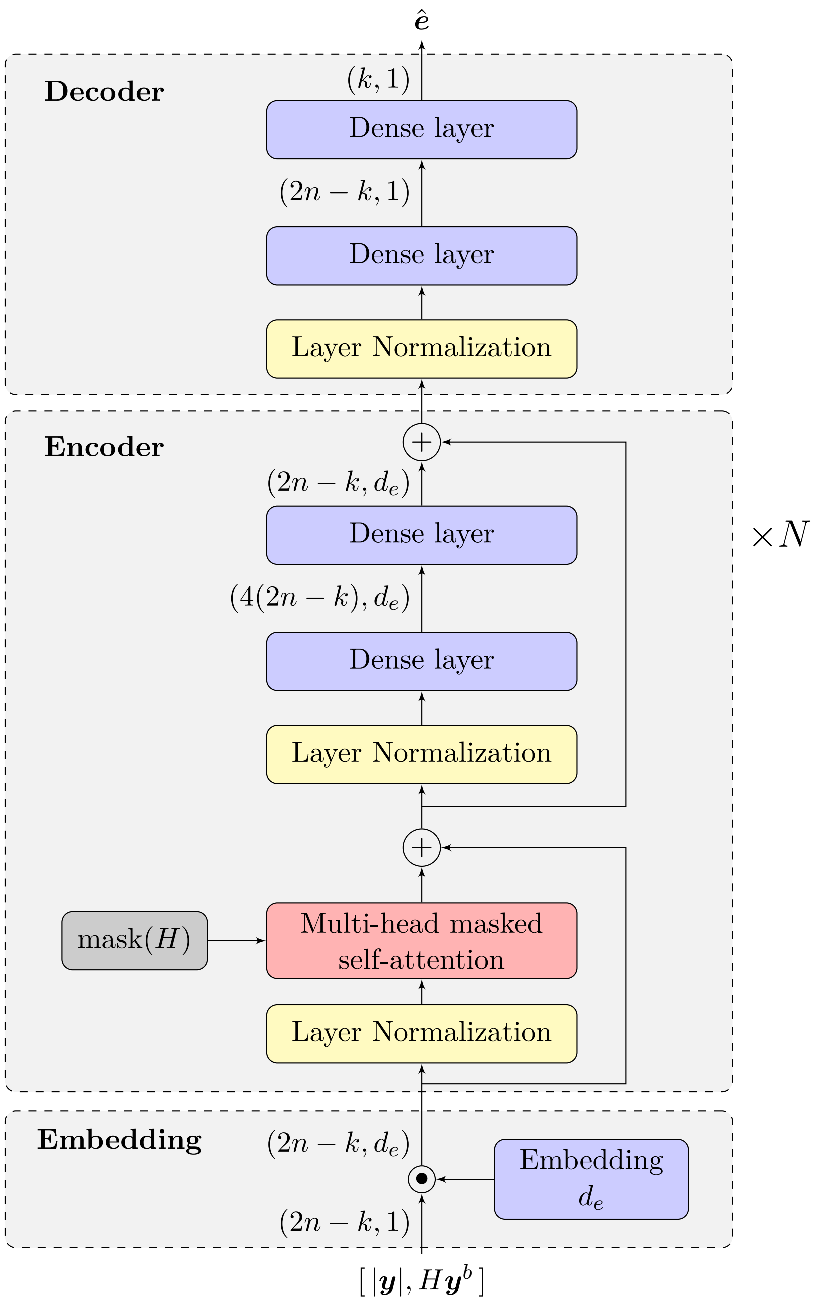


Figure 3.6: Transformer-based architecture for the noise estimator, as proposed by [CW22a].

the learning and complexity aspects of the neural component of the decoder are not well contemplated or studied. This section presents various contributions from our work in [DB23] and [DeB+24b]:

(i) We introduce a Syndrome-Based Neural Decoder (SBND) framework that extends the work of Bennatan et al. [BCK18a] to a full decoder of the message $\boldsymbol{u}$, which is directly applicable to systematic and non-systematic codes.

(ii) We present a novel recurrent version of the transformer architecture to further reduce the number of weights.

(iii) We study the influence of the parity-check matrix employing information theory-related metrics.

(iv) We offer a thorough complexity analysis that compares the considered architectures.

3.2.1 Proposed message-oriented framework

For the first contribution, let us define a new measure of error that assesses messages instead of codewords. Let pinv$(\cdot)$ define a pseudo-inverse function dependent on the linear block code, defined as an operation that transforms every valid codeword $\boldsymbol{c} \in \mathcal{C}$ to its corresponding message $\boldsymbol{u}$. That is, if $\boldsymbol{c} = G^T\boldsymbol{u}$, then:

$$\boldsymbol{u} = \operatorname{pinv}(\boldsymbol{c}). \tag{3.17}$$

Observe that, given a pseudo-inverse pinv$(\cdot)$, its application to a non-valid codeword may yield an unpredictable result. Observe also that, for the case of systematic codes, a simple pseudo-inverse consists in the extraction of its systematic bits, i.e., pinv$(\boldsymbol{c}) = A\boldsymbol{c}$ where:

$$A = \left[\, I_k \,|\, \mathbf{0}_{k,n-k} \,\right]. \tag{3.18}$$

Now, let $\tilde{\boldsymbol{u}}$ denote an artificial variable —also referred to as *noisy* message— defined by:

$$\tilde{\boldsymbol{u}} \triangleq \operatorname{pinv}(\boldsymbol{y}^b), \tag{3.19}$$

where $\boldsymbol{y}^b$ represents the hard decisions of the received BPSK signal $\boldsymbol{y}$, which may have suffered bitflips during transmissions (i.e., changes in its sign). Let $\boldsymbol{e}^b_{\boldsymbol{u}}$ denote the error vector between the original message $\boldsymbol{u}$ and $\tilde{\boldsymbol{u}}$, i.e., the bitflip pattern:

$$\boldsymbol{e}^b_{\boldsymbol{u}} \triangleq \tilde{\boldsymbol{u}} \oplus \boldsymbol{u}. \tag{3.20}$$

Observe that the so-called noisy message is not actually a signal we receive, but rather the output of a hard-decision decoder that thresholds the vector $\boldsymbol{y}$ to obtain $\boldsymbol{y}^b$ and then inverts it through the function pinv$(\cdot)$ (recall that $\boldsymbol{e}^b = \boldsymbol{c} \oplus \boldsymbol{y}^b$).

The proposed decoding framework is outlined in Figure 3.7. In brief, the estimator uses the same inputs as [BCK18a], but outputs a vector that indicates the positions of bitflips in the artificial vector $\tilde{\boldsymbol{u}}$, which will be corrected in the final stage to obtain the estimate $\hat{\boldsymbol{u}}$. This reduces the output size from n to k —which can be around half in a typical case— and allows the training algorithm to focus only on information bits, potentially sacrificing performance on the parity bits. Additionally, because the decoder will be extended to higher-order modulations in Chapter 4, we remove the sign operations from the system and work either with real numbers ($\boldsymbol{y}$, $|\boldsymbol{y}|$ and $\hat{\boldsymbol{e}}_{\boldsymbol{u}}$) or binary (the remaining vectors of Figure 3.7). To this purpose, we replace the last layer's activation function from a hyperbolic tangent to a linear activation.

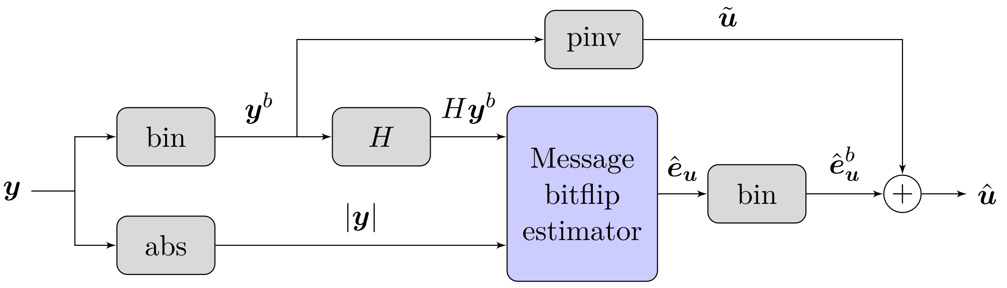


Figure 3.7: Proposed message-oriented SBND.

Consider now the following theorem.

Theorem 3.2. *Considering the previous decoder structure for estimating the message $\boldsymbol{u}$, the following equation holds:*

$$P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) = P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,H\boldsymbol{Z}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b), \tag{3.21}$$

where $\boldsymbol{Z}$ denotes the multiplicative noise from Section 3.1.2, and $\boldsymbol{e}^b_{\boldsymbol{u}} = \boldsymbol{u} \oplus \tilde{\boldsymbol{u}} = \boldsymbol{u} \oplus pinv(\boldsymbol{y}^b)$. Hence, it follows for all $i \in [1:n]$:

$$P_{U_i|\boldsymbol{Y}}(u|\boldsymbol{y}) = P_{E^b_{u,i}|\,|\boldsymbol{Z}|,H\boldsymbol{Z}^b}(u_i \oplus \tilde{u}_i \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b). \tag{3.22}$$

Proof. See Appendix A.1. □

This indicates that knowing $\boldsymbol{y}$ and computing the probability distribution of $\boldsymbol{u}$ is equivalent to knowing $H\boldsymbol{y}^b$ and $|\boldsymbol{y}|$ and computing the probability distribution of the random variable $\boldsymbol{e}^b_{\boldsymbol{u}}$, which after the final bitflip estimation is used to yield an estimate of $\boldsymbol{u}$ by applying an exclusive OR operation with the artificial variable $\tilde{\boldsymbol{u}}$:

$$\hat{\boldsymbol{u}} = \tilde{\boldsymbol{u}} \oplus \hat{\boldsymbol{e}}^b_{\boldsymbol{u}}. \tag{3.23}$$

This extends the previous results [BCK18a; CW22a; CW23] to a full decoder architecture, where the output is the estimate of the original message $\boldsymbol{u}$, and is independent of the generator matrix —and particularly, whether it is systematic or not. Observe also that Theorem 3.2 implies that the posterior distribution $P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y})$ depends only on the multiplicative noise $\boldsymbol{z}$ and is invariant with respect to the transmitted codeword. This enables single-codeword training, as long as the noise remains random throughout the learning process.

We have taken this analysis one step further by explicitly computing the expression for the argument that maximizes the posterior distribution $P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y})$. Consider the following lemma.

Lemma 3.1. *Considering the previous decoder structure for estimating the message* $\boldsymbol{u}$*, the following equation holds:*

$$P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{e}^b_{\boldsymbol{u}}} \sum_{i=1}^{n} y_i(1 - 2[G^T \text{pinv}(\boldsymbol{y}^b)]_i)[G^T \boldsymbol{e}^b_{\boldsymbol{u}}]_i, \tag{3.24}$$

where the notation $[\,\cdot\,]_i$ *indicates the i-th element of the vector between the square brackets.*

Proof. See Appendix A.2. □

This result, containing an explicit expression of the posterior distribution $P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y})$, illustrates more clearly the dependencies and complexity of the decoding process. The optimal estimate of $\hat{\mathbf{e}}^b_{\boldsymbol{u}}$ is the one that minimizes the sum of reliabilities associated with the positions of estimated bitflips, with negative values in positions where the received signal $\boldsymbol{y}$ and the encoded and modulated pseudo-inverse $(1 - 2G^T \text{pinv}(\boldsymbol{y}^b))$ have different signs and positive values otherwise. A similar procedure can be found in other decoding algorithms, such as the OSD (see Section 1.4.1) and the Chase algorithm [Cha72]. In our case, instead of computing this minimum with an iterative trial-and-testing procedure (which is costly and has high latency), we employ neural networks.

3.2.1.1 MAP denoising and MAP decoding

Let us observe that the proposed system is based on a *denoising* approach, where the estimator does not output the estimated message $\hat{\boldsymbol{u}}$ directly, but rather an estimate of the artificial quantity $\boldsymbol{e}^b_{\boldsymbol{u}}$, which is subsequently used to correct the noisy message $\tilde{\boldsymbol{u}}$. The quantity $\boldsymbol{e}^b_{\boldsymbol{u}}$ can be thought of as *message noise*, much like the codeword noise $\boldsymbol{e}^b$ in Section 3.1.3. Nonetheless, this approach is equivalent to estimating the message $\boldsymbol{u}$, as shown in the following result:

$$\operatorname*{argmax}_{\boldsymbol{u}\in\{0,1\}^k} P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) = \operatorname*{argmax}_{\boldsymbol{e}^b_{\boldsymbol{u}}\in\{0,1\}^k} P_{\tilde{\boldsymbol{U}}\oplus\boldsymbol{E}^b_{\boldsymbol{u}}|\boldsymbol{Y}}(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}}|\boldsymbol{y}) \tag{3.25}$$

$$= \operatorname*{argmax}_{\boldsymbol{e}^b_{\boldsymbol{u}}\in\{0,1\}^k} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\boldsymbol{Y}}(\boldsymbol{e}^b_{\boldsymbol{u}}|\boldsymbol{y}), \tag{3.26}$$

where we have use that $\boldsymbol{u} = \tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}}$, along with the fact that $\tilde{\boldsymbol{u}}$ is a deterministic function of $\boldsymbol{y}$ as per (3.17). Similarly, one can write that

$$\operatorname*{argmax}_{u_i\in\{0,1\}} P_{U_i|\boldsymbol{Y}}(u_i|\boldsymbol{y}) = \operatorname*{argmax}_{e^b_{u,i}\in\{0,1\}} P_{E^b_{u,i}|\boldsymbol{Y}}(e^b_{u,i}|\boldsymbol{y}), \tag{3.27}$$

which allows the transformation of the MAP decoding criterion in (1.36) to a MAP denoising criterion.

3.2.2 Recurrent transformer-based architecture

It has been stated that a central obstacle in contemporary NN-based channel decoding is the problem of *scalability*. This concept encompasses two key dimensions: (i) the network's capacity to decode larger codes and (ii) its ability to achieve this without resorting to an ever-increasing number of model parameters. The work in [CW22b] tackled this problem, achieving slightly better performances than [BCK18a] for a BCH code of size $(127, 64)$ with only a tenth of the number of weights.

As we will see in Section 3.2.3, the total number of weights in the ECCT increases predominantly as Nd_e^2, where N indicates the number of encoders and d_e the embedding diemension. The embedding provides the system with the freedom to represent the input vector in a high-dimensional space and thereby learn the output accordingly. For this reason, we decided to address the issue of the number of encoders N. In this section, we propose a recurrent version of the ECCT —henceforth referred to as r-ECCT— that takes a step in the scalability direction. As evidenced in Section 3.2.3, the transformer solution has roughly a linear dependency on the number of concatenated attention blocks (i.e., encoders). We propose removing this dependency by adopting a recurrent strategy, where the same attention block is employed iteratively for N cycles. The basic architecture is outlined in Figure 3.8: after the embedding phase, the data is fed to the encoder block that iterates on itself N times. The resulting encoded vector of size $(2n - k, d_e)$ is fed to the decoder to estimate the bitflip vector.

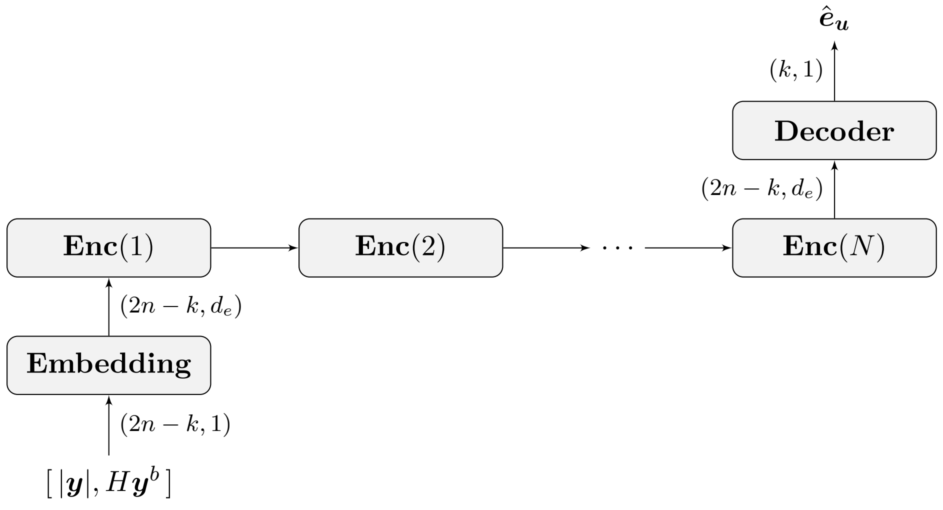


Figure 3.8: r-ECCT solution proposed for the bitflip estimator.

3.2.3 Complexity analysis

This section presents the complexity involved in the four considered NN architectures regarding the number of parameters that configure the network as a function of certain hyperparameters to be selected in each case. Observe that the message-oriented approach of Section

3.2.1 is adopted, but with the exception of the output layer, the complexity analysis is also applicable to the decoding framework of Section 3.1.3.

3.2.3.1 Multi-Layer Perceptron (Feed-Forward Dense Network)

For a dense layer with n_i inputs and n_o outputs (i.e., n_o neurons), the number of weights is given by $n_i n_o + n_o$, which includes the multiplicative weights and the biases. Let r denote the length of the input vector $[\,|\boldsymbol{y}|, \boldsymbol{H}\boldsymbol{y}^b\,]$, i.e., $r \triangleq (2n-k)$, and $\alpha \in \mathbb{N}^+$ a scaling hyperparameter so that each layer contains $\alpha(n-k)$ units. Then, for an MLP consisting of d_l hidden layers —plus the output layer—, the number of weights is given by:

$$\begin{aligned} \mathbb{W}_{\text{MLP}} &= \underbrace{\alpha r^2 + \alpha r}_{\text{first layer}} + \underbrace{d_l((r+\alpha r)\alpha r + \alpha r)}_{d_l - 1 \text{ hidden layers}} + \underbrace{(r+\alpha r)k + k}_{\text{output layer}} \\ &= (d_l\alpha + d_l + 1)\alpha r^2 + (d_l+1)\alpha r + (r+\alpha r)k + k, \end{aligned} \tag{3.28}$$

which, for a fixed depth d_l and scaling parameter α, implies a network size that increases roughly as $\mathcal{O}(r^2)$.

3.2.3.2 Recurrent Neural Networks

The RNN implemented in our work and in [BCK18a] is based on GRU cells. If n_i and n_o represent the dimensions of the input and the output, respectively, we have that the total number of weights for a GRU equals $3(n_o^2 + n_i n_o + n_o)$ [DS17]. Moreover, the number of parameters for the final dense layer is given by $n_i n_o + n_o$. Recalling that each GRU cell consists of $\alpha(2n-k)$ GRUs, and that the network contains d_l GRU cells stacked on top of each other, we have that the total number of weights for the RNN is given by:

$$\mathbb{W}_{\text{RNN}} = \underbrace{3(\alpha^2 r^2 + \alpha r^2 + \alpha r)}_{\text{input GRU layer}} + \underbrace{(d_l - 1)3(\alpha^2 r^2 + \alpha^2 r^2 + \alpha r)}_{d_l - 1 \text{ hidden GRU layers}} + \underbrace{\alpha r k + k}_{\text{dense layer}}, \tag{3.29}$$

where $r \triangleq 2n-k$ is the size of the network's input and (n,k) represents the parameters of the channel code. Collecting the terms, the following result is obtained:

$$\mathbb{W}_{\text{RNN}} = 3\Big((2d_l - 1)\alpha^2 + \alpha\Big) r^2 + (3d_l + k)\alpha r + k. \tag{3.30}$$

It is worth observing that, for a fixed depth d_l and a scaling parameter α, this result implies a network that increases in size as $\mathcal{O}(r^2)$.

3.2.3.3 Error Correction Code Transformer

The transformer-based architecture is composed of three stages: (i) the embedding, which involves rd_e weights, with d_e the embedding dimension; (ii) the N encoders, each composed of two normalization layers, an MH-SA, and two dense layers; and (iii) the decoder, consisting of a normalization layer and two dense layers. Hence, the number of weights can be expressed as follows:

$$\begin{aligned} \mathbb{W}_{\text{ECCT}} = N\big(\underbrace{4d_e}_{\text{norm.}} + \underbrace{4d_e(d_e+1)}_{\text{MH-SA}} + \underbrace{4d_e^2 + 4d_e + 4d_e^2 + d_e}_{\text{encoder dense layers}} \big) \\ + \underbrace{2d_e}_{\text{norm.}} + \underbrace{d+1+rk+k}_{\text{decoder dense layers}} + \underbrace{rd_e}_{\text{embedding}} \,. \end{aligned} \tag{3.31}$$

Once again, collecting the terms, we get the following expression for the total number of weights in the ECCT:

$$\mathbb{W}_{\text{ECCT}} = 12Nd_e^2 + (13N + r + 3)d_e + (r+1)k + 1, \tag{3.32}$$

where, for a fixed number of encoders and embedding dimension, the value of $\mathbb{W}_{\text{ECCT}}$ is almost independent of the code parameters (n, k). Its main dependence is on the number of encoders N (linear) and the dimension of the embedding d_e (quadratic), and the total number of parameters increases roughly as $\mathcal{O}(Nd_e^2)$.

3.2.3.4 Recurrent ECCT

The proposed solution is rooted in the ECCT architecture, with a novel adaptation involving the recurrent utilization of the encoder for the N iterations. Hence, the number of parameters can be easily obtained by replacing $N = 1$ in (3.32):

$$\mathbb{W}_{\text{r-ECCT}} = 12d_e^2 + (16 + r)d_e + (r+1)k + 1. \tag{3.33}$$

The proposed NN is virtually only dependent on the embedding dimension, which is considered fixed in this work. Observe that, in practice, larger codes will probably require larger embedding dimensions to maintain competitive decoding performances.

3.2.4 Influence of the parity-check matrix

Set aside the hyperparameters of the NN, the performance of the overall decoder turns out to be very dependent on the parity check matrix H. In the following, we study the influence of this matrix H on the proposed decoder. We propose an information theory-based metric that evaluates the information shared between the syndrome input $H\boldsymbol{y}^b$ and the bitflip vector $\boldsymbol{e}^b$ as a function of the employed parity-check matrix. We then suggest an algorithm to sparsify the parity-check matrix, which in turn increases the proposed metric. Finally, we evaluate the impact of this new matrix by simulating the decoder with different parity check matrices and displaying the resulting BER. For simplicity, the analysis is carried out employing the codeword bitflip vector $\boldsymbol{e}^b$.

3.2.4.1 Sparsifying the parity-check matrix

In the model-free approach, one of the inputs to the noise estimator is the syndrome $\boldsymbol{s} \triangleq H\boldsymbol{y}^b$. The learning capabilities of the neural estimator may vary according to the choice of the parity-check matrix, e.g., its shape, structure, or density. This section presents a way to measure the impact of the parity-check matrix on the learning capabilities of the network and proposes a method to construct a matrix that improves performance when needed. First, let us recall the following definition from [Sha48].

Definition 3.1 (Mutual Information)**.** Let X and Y denote two discrete random variables with joint probability mass function $P_{X,Y}$. Then the Mutual Information (MI) between X and Y is defined by:

$$I(X;Y) \triangleq \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P_{X,Y}(x,y)\log_2\left(\frac{P_{X,Y}(x,y)}{P_X(x)P_Y(y)}\right), \tag{3.34}$$

where P_X and P_Y denote the marginal probability mass functions of X and Y associated with $P_{X,Y}$.

Consider now the following theorem.

Theorem 3.3 (MI and parity-check matrix)**.** *Let $\mathcal{C}$ be a code defined by a generator matrix G and $\boldsymbol{E}^b$ a binary random variable that represents the bitflip pattern induced by the channel. Let H and $\tilde{H}$ denote any two valid parity-check matrices for the code $\mathcal{C}$. Then:*

$$I(\boldsymbol{E}^b;\boldsymbol{S}) = I(\boldsymbol{E}^b;\tilde{\boldsymbol{S}}), \tag{3.35}$$

where $\boldsymbol{S} \triangleq H\boldsymbol{E}^b$ and $\tilde{\boldsymbol{S}} \triangleq \tilde{H}\boldsymbol{E}^b$ represent the syndromes induced by each parity-check matrix.

Proof. See Appendix B.1 □

This result implies that changing the parity-check matrix H induces no intrinsic loss in the shared information between the overall syndrome $\boldsymbol{S}$ and the overall bitflip vector $\boldsymbol{E}^b$. However, when taking into account the architecture of the NN and the formulation of the training loss function, it appears more intuitive to analyze the impact on each of the components E_i^b separately. In this work, we choose to investigate a *surrogate* information measure defined in the following lemma.

Lemma 3.2 (Pairwise MI). *In the same framework as in Lemma 3.3, if $P_{E_i^b}(1) = p$ for all $i \in [1 : n]$, then the MI between the ith component of the bitflip vector $\boldsymbol{E}^b$ and the jth component of the syndrome $\boldsymbol{S}$ is given by:*

$$I(E_i^b; S_j) = [\mathcal{H}_b(\mathcal{E}(N_j)) - \mathcal{H}_b(\mathcal{E}(N_j - 1))] \mathbb{1}(H_{ij} = 1), \tag{3.36}$$

where:

- *$\mathcal{H}_b(a) \triangleq -a log_2 a - (1-a) log_2(1-a)$ is the binary entropy function;*
- *N_j denotes the Hamming weight of the jth row of the parity-check matrix H;*
- *and $\mathcal{E}(N_j)$ denotes the probability of obtaining an even number of positive outcomes after N_j Bernoulli realizations with probability p. That is:*

$$\mathcal{E}(N_j) \triangleq P\{even \ \# \ of \ ones\} = \frac{1}{2} + \frac{1}{2}(1-2p)^{N_j}. \tag{3.37}$$

Proof. See Appendix B.2. □

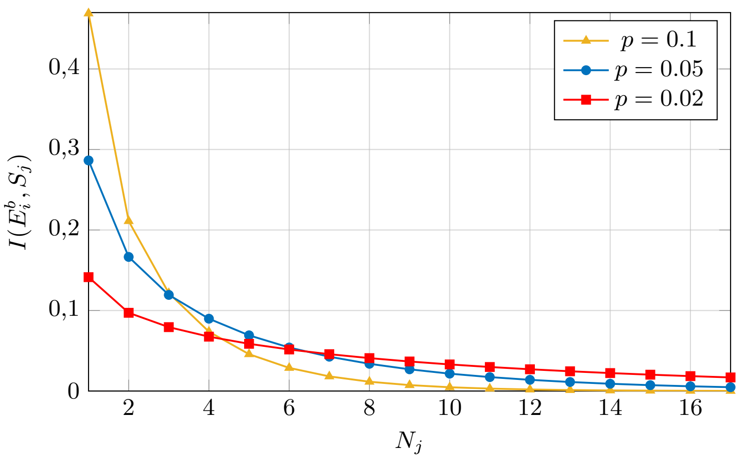


Figure 3.9: Mutual information between E_i^b and S_j as a function of the weight of the jth row N_j, for different bitflip probabilities $p \in \{0.02, 0.05, 0.1\}$.

The quantity $I(E_i^b; S_j)$ is represented as a function of N_j in Figure 3.9. The shared information between E_i^b and S_j decreases with the weight of the jth row, and reaches its maximum when $N_j = 1$. This indicates that, even though the quantity $I(\boldsymbol{E}^b; \boldsymbol{S})$ is independent of the choice of the parity-check matrix, the pairwise mutual information $I(E_i^b; S_j)$ depends inversely on the weight of the corresponding row.

Algorithm 2 2nd-order parity-check matrix sparsifying

```
Input: parity-check matrix H
Output: sparsified parity-check matrix $H_r$
1: $H_r \leftarrow H$
2: FoundBetter ← True
3: $C = \{\{i\} : 1 \leq i \leq n\} \cup \{\{i, j\} : 1 \leq i < j \leq n\}$
4: while FoundBetter do
5:   $\omega_{\text{min}} \leftarrow n$
6:   FoundBetter ← False
7:   for $i \in [1, 2, ..., n - k]$ do
8:     $\omega_i \leftarrow$ weight(H[$i$])
9:     for comb $\in C \,/\, i \notin$ comb do
10:      $\omega_{\text{new}} \leftarrow$ weight($\bigoplus$($H_r[i]$, $H_r$[comb]))
11:      if $\omega_{\text{new}} < \min(\omega_{\text{min}}, \omega_i)$ then
12:        $\omega_{\text{min}} \leftarrow \omega_{\text{new}}$
13:        $(i_{\text{best}}, \text{comb}_{\text{best}}) \leftarrow (i, \text{comb})$
14:        FoundBetter ← True
15:      end if
16:    end for
17:  end for
18:  if FoundBetter then
19:    $H_r[i] \leftarrow \bigoplus(H_r[i_{\text{best}}], H_r[\text{comb}_{\text{best}}])$
20:  end if
21: end while
22: return $H_r$
```

In order to increase the pairwise mutual information $I(E_i^b; S_j)$, Algorithm 2 outlines a simple procedure to sparsify a matrix without changing its kernel space, i.e., applying exclusively linear operations among its rows. The algorithm exhaustively searches all the pairs and triples of rows, assessing which addition of rows leads to the largest reduction in the matrix's overall weight. This process is repeated until no combination of two or three rows reduces the weight of the parity-check matrix.

We can now define the Mean Pairwise Mutual Information (MPMI) as follows:

$$\text{MPMI}(E_i^b, \boldsymbol{S}) = \frac{1}{n-k} \sum_{j=1}^{n-k} I(E_i^b; S_j). \tag{3.38}$$

If this quantity is larger, then on average, more information is shared between each E_i^b and S_j. For a systematic BCH code with $n = 127$ and $k = 64$, where the first k positions correspond to the information bits, let us consider four different possible parity-check matrices depicted in Figure 3.10:

1. The *original* parity-check matrix, taken from that channel code database of [Hel+19];

2. its standardized version, as per (1.12);
3. a randomized parity-check matrix, computed by applying several random linear combinations over the original matrix's rows and;
4. a sparsified matrix, obtained by applying Algorithm 2 to the standardized matrix.

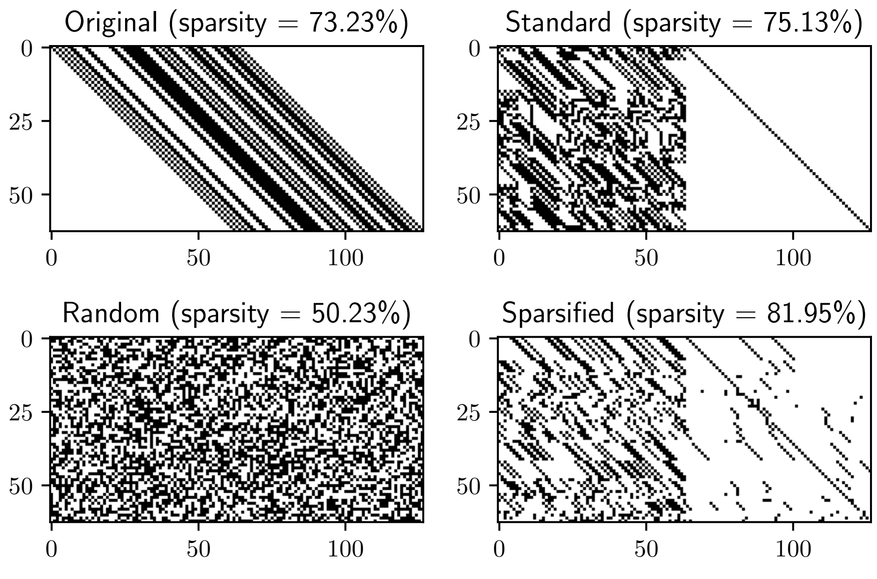


Figure 3.10: Possible parity-check matrices for a BCH code of size $(127, 64)$. Ones are represented in black and zeros in white. Their sparsity (percentage of zeros in the matrix) is also displayed.

Figure 3.11 shows the MPMI between the syndrome $\boldsymbol{S}$ and each bitflip variable E_i^b. Among the four considered parity-check matrices, the sparsified matrix stands out for two main reasons: (i) its MPMI is on average the highest for most bitflip positions and (ii) because systematic codes are employed, the indices for which the MPMI is lower correspond to the redundancy bits that the decoder will not estimate. Intuitively, the sparsified matrix is the one that has the highest average information shared between the syndrome (i.e., the decoder's input that depends on the parity-check matrix) and the systematic bitflip components (i.e., the decoder's output in the message-oriented approach).

3.3 Experiments

This section presents the decoding performance of our proposed decoder from Section 3.2.1 over different codes, namely BCH and Polar codes of different sizes and code rates. To recapitulate, our contributions can be divided as follows:

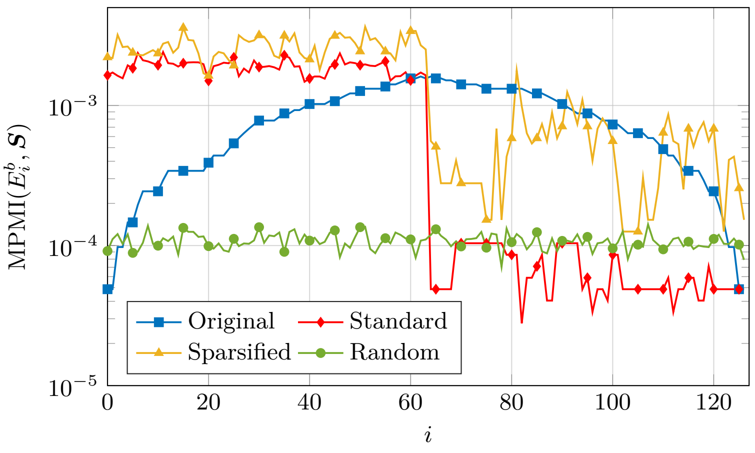


Figure 3.11: MPMI between the syndrome $\boldsymbol{S}$ and each bitflip component E_i^b.

(i) The proposed decoding framework from Section 3.2.1 —denoted *message-oriented*— that puts its focus on information bits (i.e., the message $\boldsymbol{u}$);

(ii) the recurrent transformer architecture that removes the dependency on the number of encoders, thus reducing the number of parameters by a factor of N;

(iii) and the parity-check matrix analysis, with its subsequent sparsified version that improves training and performance metrics.

To demonstrate the interest of all of these elements, they are implemented sequentially. The model-free decoders from [BCK18a] and [CW22a] provide the best performances found in the state-of-the-art. Hence, we start by implementing these decoders and comparing them to the same architectures but using our framework from Section 3.2.1. Once our framework has been justified, it is retained and used for the remaining simulations. Subsequently, the r-ECCT from Section 3.2.2 is compared to the previous architectures, displaying competitive performances while using only a fraction of the previous networks' number of parameters. Finally, the parity-check matrix is considered, selecting the sparsified matrix as per Section 3.2.4. A final simulation is provided to aggregate all of the contributions and compare the resulting decoder against the systems in [BCK18a; CW22a].

3.3.1 Training and hyperparameters

To test the decoders and our different contributions, the system from Section 3.1.1 is simulated, employing the decoders from Sections 3.1.3 and 3.2.1 and the classical decoders OSD (for the BCH codes) and SCL (for the Polar codes). We implemented all the NN-based decoder architectures using Google's TensorFlow library [Mar+15] along with the Keras API [Cho+15]. Binary data is generated *on the fly* and grouped into batches of varying size, depending on the architecture and training limitations. The binary cross-entropy function is employed as the loss function during training, defined for two vectors $\boldsymbol{a} = \{a_i\}_{1\leq i\leq k}$ and

$\hat{\boldsymbol{a}} = \{\hat{a}_i\}_{1 \leq i \leq k}$ as follows:

$$\mathcal{L}_{\text{BCE}}(\boldsymbol{a}, \hat{\boldsymbol{a}}) = \sum_{i=1}^{k} \Big(a_i \log(\hat{a}_i) + (1 - a_i) \log(1 - \hat{a}_i) \Big), \tag{3.39}$$

where $\boldsymbol{a}$ is conventionally the true value (or *ground truth*) and $\hat{\boldsymbol{a}}$ denotes the NN-estimated value. The last dense layer of every architecture was selected to be linear, as to *mimic* LLR values. As such, a sigmoid function is applied to the estimators' outputs before computing the binary cross-entropy loss. Another possibility would be to select the sigmoid as the final activation function and use a $1/2$-threshold to convert to the binary domain. To estimate the BER for a given E_b/N_0, a Monte Carlo simulation is carried out, with a stopping criterion of 500 frame errors and with a minimum of 10^4 frames sent. The BER is computed message-wise, meaning that the comparison is carried out between the transmitted message $\boldsymbol{u}$ and the estimated one $\hat{\boldsymbol{u}}$.

Observation 3.2. When employing the codeword-oriented decoder —which gives an estimate of the transmitted codeword $\boldsymbol{c}$—, an inverse is applied to obtain an estimation of the message $\boldsymbol{u}$. In the case of BCH codes, we employ their systematic version and thus the inverse is easily obtained by an extraction of the information bits. For Polar codes, we make use of Lemma 1.1.

Details about each simulation scenario are provided gradually as new elements are added to the system. For reproducibility, the architecture and training hyperparameters employed are summarized in Table 3.1, and the total number of weights in each architecture is displayed in Table 3.2. All the simulated codes are obtained from the channel code database of [Hel+19]. To maintain a variety in the code size and rate without incurring an overload of figures and information, different codes will be employed for each contribution. Nevertheless, tendencies are very similar regardless of which of the considered codes is selected for each section.

3.3.2 Simulation results

3.3.2.1 Previous architectures

Because we implemented the decoders ourselves, let us start by benchmarking the three NN-based architectures employed in the literature: MLP, RNN, and ECCT. For this, we selected two BCH codes of size $(63, 45)$ and $(127, 64)$ as they are implemented in both [BCK18a] and [CW22a], and because they represent very different outcome scenarios.

Figure 3.12a shows the BER performance of the three architectures, along with the ML curve computed using an OSD of order 3. We can observe that, while the RNN (4.5M weights) reaches pseudo-optimal performance, the ECCT (2M weights) and MLP (3.7M weights) present a gap of approximately 1dB. Figure 3.12b exhibits a very different behavior, where even the RNN (20M weights) is unable to beat even the OSD of order 1, even if the gap begins to close in higher SNR. However, in this case, the ECCT shows its interest, where

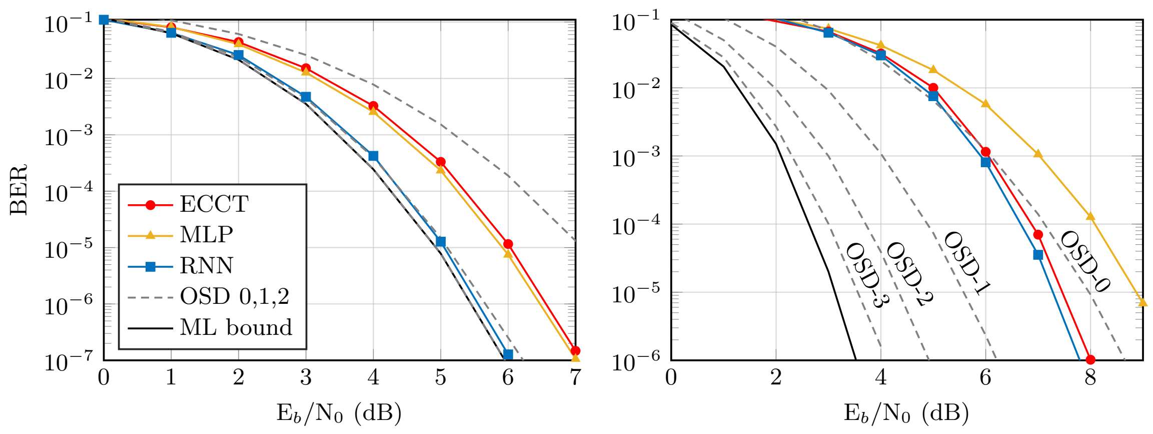
Code	Rate	Training E_b/N_0	MLP	RNN	ECCT	r-ECCT
BCH $(63, 57)$	0.90	4dB	$\alpha = 5$, $d_l = 5$, b.s. $= 2^{12}$	$\alpha = 7$, $d_l = 10$, b.s. $= 2^{12}$	$d_e = 128$, $N = 10$, b.s. $= 2^8$	$d_e = 128$, $N = 10$, b.s. $= 2^9$
BCH $(63, 51)$	0.81					
BCH $(63, 45)$	0.71	3dB				
BCH $(63, 39)$	0.62					
BCH $(127, 64)$	0.50	4dB				
Polar $(64, 48)$	0.75	3dB	$\alpha = 7$, $d_l = 10$, b.s. $= 2^{12}$	$\alpha = 5$, $d_l = 5$, b.s. $= 2^{12}$	$d_e = 128$, $N = 10$, b.s. $= 2^8$	$d_e = 128$, $N = 10$, b.s. $= 2^9$
Polar $(64, 32)$	0.50					
Polar $(64, 22)$	0.34					
Polar $(128, 96)$	0.75	4dB				
Polar $(128, 64)$	0.50					

Table 3.1: Summary of the hyperparameters used in each simulated scenario, for each NN architecture.

Code	$\mathrm{W}_{\mathrm{MLP}}$	$\mathrm{W}_{\mathrm{RNN}}$	$\mathrm{W}_{\mathrm{ECCT}}$	$\mathrm{W}_{\text{r-ECCT}}$
BCH $(63, 57)$	2.7M	2.6M	2M	200k
BCH $(63, 51)$	3.2M	3.1M		
BCH $(63, 45)$	3.8M	3.6M		
BCH $(63, 39)$	4.3M	4.1M		
BCH $(127, 64)$	20.5M	19.5M		
Polar$(64, 48)$	3.7M	3.5M	2M	200k
Polar $(64, 32)$	5.3M	5M		
Polar $(64, 22)$	6.4M	6.1M		
Polar $(128, 96)$	14.6	13.9M		
Polar $(128, 64)$	21M	20M		

Table 3.2: Summary of the approximate number of weights that configure each NN architecture.

with only 2M parameters, it greatly outperforms the MLP decoder (which has 20.5M weights). Lastly, it is worth observing that all the performances are as expected from [BCK18a] and [CW22a], with the exception of the RNN-based decoder for the BCH$(127, 64)$, for which our simulations displayed slightly better results than those shown in [BCK18a].



(a) BCH (63,45).

(b) BCH (127,64).

Figure 3.12: Bit error rate comparison for two BCH code of sizes $(63, 45)$ and $(127, 64)$ under a BPSK modulation scheme, employing the three considered architectures from the state-of-the-art. The MLB curve was computed using an OSD of order 3.

3.3.2.2 Message-oriented vs. codeword-oriented decoders

In the following, our proposed message-oriented approach is compared to the previous decoder of Section 3.1.3, which we have denominated, in opposition, codeword-oriented. Recall that the interest of our proposed decoder lies mainly in two aspects: (i) its focus on information bits rather than parity bits, seeking to minimize the information BER rather than the overall codeword BER; (ii) its one-shot decoding approach that directly estimates the message, without passing through the estimated codeword and thus avoiding extra computations when non-systematic codes are employed. To assess numerically the performance gain of this approach, two codes are implemented: a BCH code of size $(127, 64)$ and a Polar code of size $(64, 32)$. The three architectures (RNN, ECCT, and MLP) are implemented for both approaches, but only two were kept: the RNN, as a high-performance and highly complex solution, and the ECCT, as the lighter but still competitive alternative —in occasions surpassing the RNN. The MLP has a moderately high complexity (close to the RNN), but with a performance that rarely surpasses that of the ECCT, and hence is left out of the following sections.

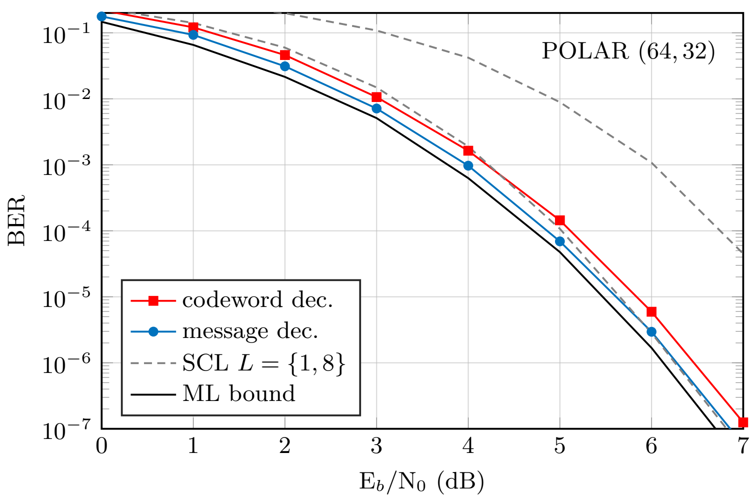
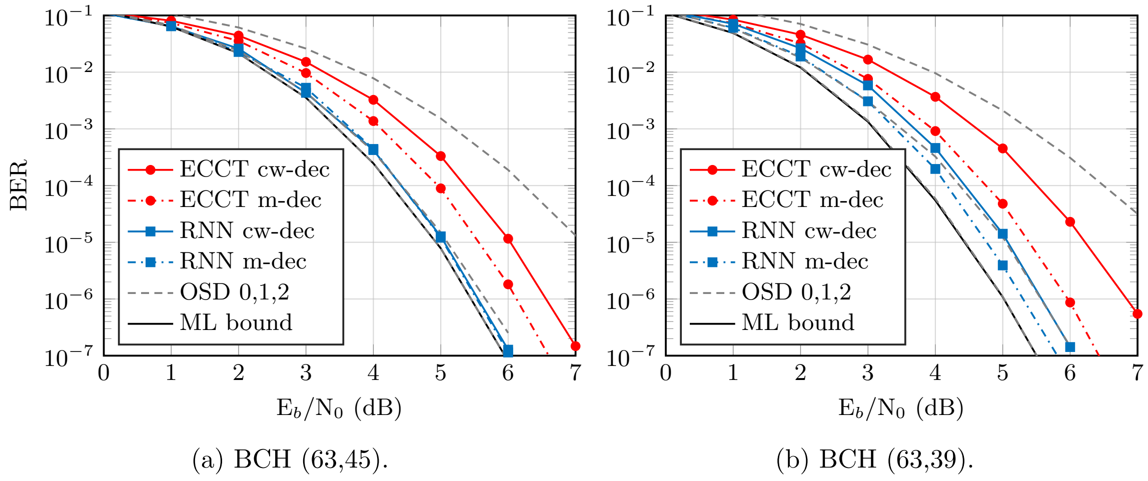


Figure 3.13: Bit error rate comparison for a Polar code of size $(64, 32)$ under a BPSK modulation scheme. Both neural decoders are implemented using RNNs.



(a) BCH (63,45). (b) BCH (63,39).

Figure 3.14: Bit error rate comparison for two BCH codes of size $n = 63$ and $k = \{45, 39\}$, under a BPSK modulation scheme, employing the codeword-oriented approach (cw-dec) and message-oriented (m-dec).

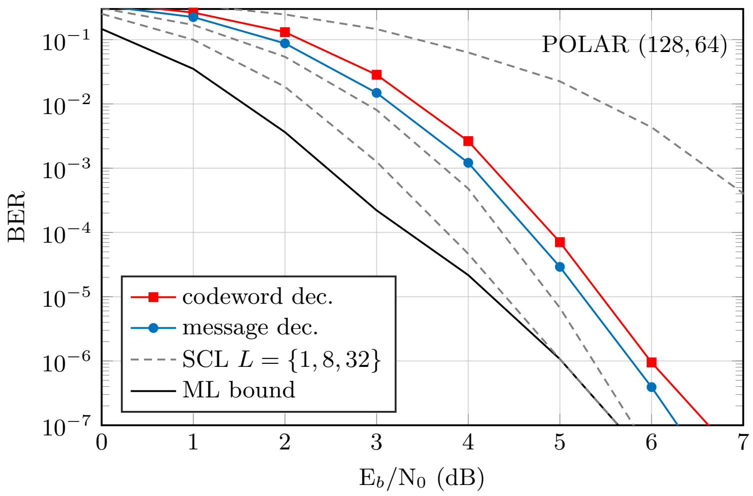


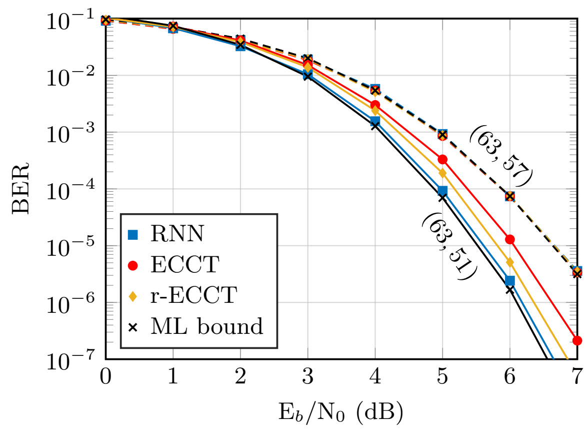
Figure 3.15: Bit error rate comparison for a Polar code of size $(128, 64)$ under a BPSK modulation scheme. Both neural decoders are implemented using the ECCT.

3.3.2.3 Recurrent ECCT architecture

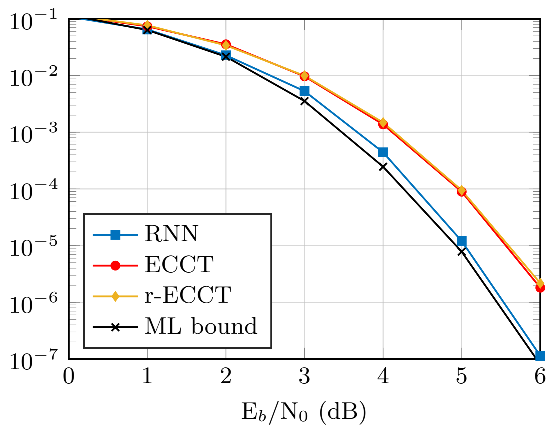
Let us now recall and evaluate the interest of the proposed r-ECCT architecture with respect to the RNN and ECCT. We have seen in Section 3.2.3 that the RNN is, by far, the most complex architecture in terms of the number of parameters needed to build the network. Additionally, this number grows exponentially with the size of the code. This motivated the application of the Transformer architecture of [Vas+17] to the model-free error correction problem, giving rise to the ECCT in the work by Choukroun et al. in 2022 [CW22a]. A fixed embedding dimension $d_e = 128$ and a number of encoders $N = 10$ —which are the values used in the original work for the largest codes— yields a network composed of approximately 2 million parameters. For this reason, we proposed in Section 3.2.2 to take advantage of the recurrent nature of the transformer-based architecture, using the same attention block throughout the N iterations. We then saw in the complexity analysis that this reduces the number of parameters by a factor equal to N, which is equal to 10 in our simulations. Hence, the resulting network has approximately 200k parameters. In Figures 3.16 and 3.17, we can see its interest, where the r-ECCT performs very similarly to the ECCT architecture with only a tenth the number of parameters.

3.3.2.4 Sparsified parity-check matrix

For the final contribution on SBND for BPSK, we employed information theory-based metrics to justify the interest of sparser versions of the parity-check matrix for the model-free decoding approach. To better showcase the performance gains of this contribution, we selected a BCH code of size $(127, 64)$ for two main reasons: (i) it is large and *difficult* enough so that the simulated decoders do not reach optimal performance (leaving room for improvement); and (ii) its parity-check matrix is dense enough so that the sparsifying algorithm reduces significantly

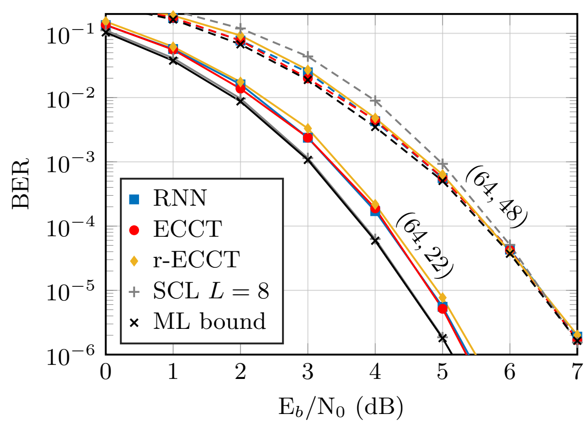


(a) BCH (63,57) and BCH (63,51).

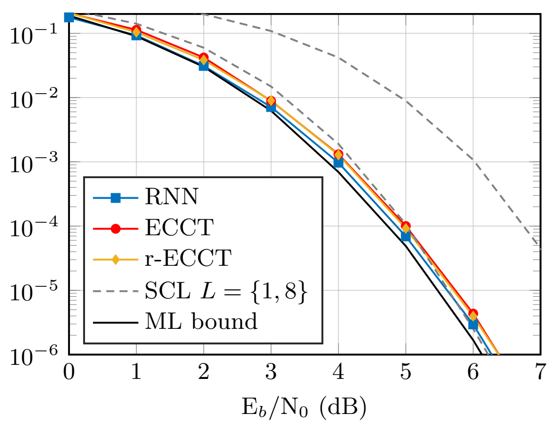


(b) BCH (63,45).

Figure 3.16: Bit error rate comparison for three BCH codes of size $n = 63$ and $k = \{57, 51, 45\}$, under a BPSK modulation scheme.



(a) Polar (64,48) and Polar (64,22).



(b) Polar (64,32).

Figure 3.17: Bit error rate comparison for three Polar codes of size $n = 64$ and $k = \{48, 32, 22\}$, under a BPSK modulation scheme.

its weight. Let us now recall the four parity-check matrices considered in Section 3.2.4:

1. The *original* parity-check matrix, taken from that channel code database of [Hel+19];
2. its standardized version, as per (1.12);
3. a randomized parity-check matrix, computed by applying several random linear combinations over the original matrix's rows and;
4. a sparsified matrix, obtained by applying Algorithm 2 to the standardized matrix.

The MPMI of each matrix is shown in Figure 3.11, where we can see that our proposed sparsified matrix has the highest MPMI in essentially every position. We train the decoders using the r-ECCT architecture and the message-oriented approach, for each of the considered parity-check matrices. Results are shown in Figure 3.18, where we can observe that the sparsified parity-check matrix shows significantly better results than the other considered matrices.

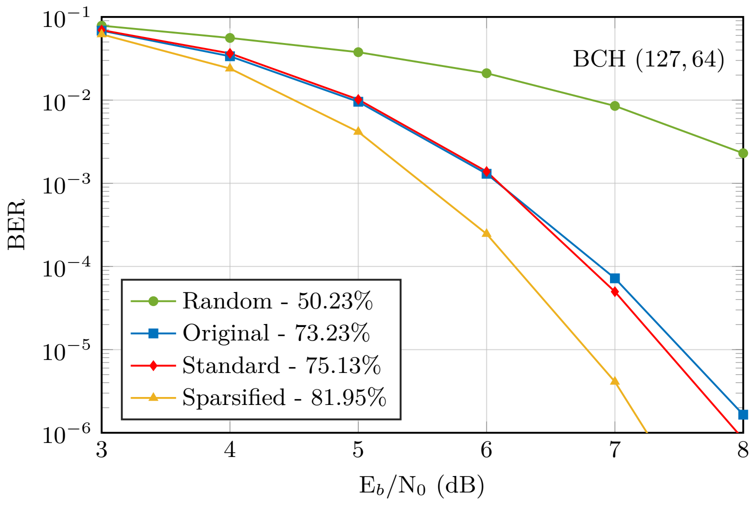


Figure 3.18: Bit error rate comparison for a BCH code of size $(127, 64)$ under a BPSK modulation scheme, employing the r-ECCT architecture and the four considered matrices. The sparsity of each matrix is also added in the legend.

Finally, Figure 3.19 shows a final comparison between previous solutions and our proposed solution, which aggregates the message-oriented approach, the r-ECCT, and the sparsified parity-check matrix. With these three elements, we are able to outperform the model-free solutions of [BCK18a] and [CW22a] —that do not employ the message-oriented approach nor optimize the parity-check matrix— while using only a small fraction of the total number of network parameters[3]. Additionally, the BER of the model-based solution from [Nac+18] is added for comparison (see Introduction), which includes a version using the regular parity-check matrix from [Hel+19], and another decoder that employs the cycle-reduced parity-check matrix based on [HC06].

[3]Better performances can be obtained if we are willing to increase the network's size.

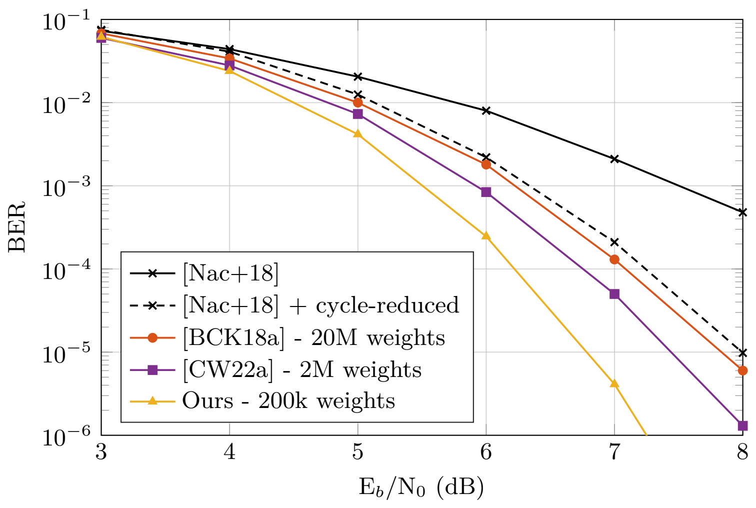


Figure 3.19: Performance comparison between our proposed solution (with all considered elements) and the previous solutions from [BCK18a; CW22a; Nac+18] for a BCH code of size $(127, 64)$ and a BPSK modulation.

Chapter 4

Application of SBND to higher-order modulations

Contents

Thus far, we have discussed scalability as the main obstacle pertaining to neural-based decoders for realistic applications. In response, the model-free approach —which we denoted as Syndrome-Based Neural Decoding (SBND)— was presented in detail in Chapter 3. Therein, we revisited the decoding framework from a previous work [BCK18a] and proposed a series of contributions that aim to increase performance while reducing the overall decoder's complexity. The main idea behind the SBND is to produce a symmetric decoder that does not depend on the codeword and can thus be trained with a unique codeword, without having to explore the entirety of a codeword space of size 2^k for a k-dimensional code. Although this provides us with a very promising framework, this system relies extensively, hitherto, on the properties of BPSK, and can be easily extended to QPSK. However, in order to allow for practical implementations, SBND has to be extended to higher-order modulations such as M-QAM and M-PSK for arbitrary M. In this Chapter, we propose an extension of the SBND that can be directly applied to such linear modulation techniques. More particularly, we focus on Bit-Interleaved Coded Modulations (BICM) [Alv08; CTB98] which, unlike classical coded modulation schemes, present the advantages of allowing the usage of any FEC code designed for memoryless channels, and of being more robust to burst errors.

In contrast to the proposed decoder of Chapter 3, which exploited the real and symmetric nature of the BPSK modulation, when employing higher-order modulations, the statistical properties of the received signal (e.g., the probability of a bitflip) vary depending on the transmitted symbol, which may be more prone to certain errors in certain positions. For this reason, BICM are adopted in order to regain the necessary symmetry properties to be able to prove the model-free decoder's optimality. We propose a decoding framework that uses the bit Log-Likelihood Ratios (bit-LLR) as input and a bit-interleaver to break the asymmetries between positions in the symbols.

The contributions in this Chapter can be classified as follows:

(i) We propose an SBND that extends the system from Chapter 3 to the case of higher-order BICM, maintaining the message-oriented approach therein presented. For this, we first characterize the equivalent channel between the transmitted codewords $\boldsymbol{c}$ and the received LLRs $\boldsymbol{l}$ under a BICM scenario, using results from the literature [Alv08; CTB98].

(ii) We prove the optimality of the proposed system, highlighting the differences and limitations with respect to the BPSK-specific decoder of Chapter 3, namely the training set design and the loss of the *single-codeword training property.*

(iii) We evaluate the performance of the proposed decoder employing the main architectures from the literature (i.e., RNN and ECCT), along with our proposed architecture from Chapter 3 (i.e., the r-ECCT).

(iv) Finally, we include a discussion on generic Coded Modulations (CM), their difference pertaining to the model-free approach, and the limitations they impose.

The main difference between higher-order modulations and BPSK/QPSK is that the decoder is not directly fed with the channel output, but rather with bit-LLRs produced by the soft demodulator as per (1.21). Hence, in order to design an SBND for the BICM setting, we first start by introducing the BICM system layout and characterizing the channel induced by the bit-LLRs for two common modulation schemes. Next, we propose an SBND that extends the system from Chapter 3 to the case of higher-order BICM, maintaining the message-oriented approach therein presented. Finally, we analyze the performance of the main considered architectures for the neural-based decoders, namely, RNN, ECCT, and r-ECCT.

The work in this Chapter can be found in [DeB+24c] and [DeB+24d].

4.1 Preliminaries and problem statement

In this section, we offer an overview of the system model, along with an equivalent model for BICM and the resulting posterior distribution, which will be employed in the following sections.

4.1.1 System model with BICM

Let us start by introducing the BICM framework depicted in Figure 4.1. As before, a random source generates binary data in frames of length k —referred to as *message* and noted by $\boldsymbol{u}$—, assumed independent and uniformly distributed. This message is then mapped to an n-bit codeword $\boldsymbol{c}$ through a linear FEC code defined by a generator matrix G of size $k \times n$, such that $\boldsymbol{c} = G^T\boldsymbol{u}$. Then, a perfectly random interleaver Π, assumed to be known to the receiver as well, shuffles the bits of $\boldsymbol{c}$ into an n-bit sequence $\tilde{\boldsymbol{c}}$. The bit-interleaved codeword is finally mapped through a linear modulation scheme of order m, such as PSK or QAM, that yields a modulated codeword $\boldsymbol{x} \in \mathbb{C}^{n'}$ of length $n' = n/m$. The channel introduces an AWGN $\boldsymbol{w} \sim \mathcal{CN}(\boldsymbol{0}, \sigma^2 I_{n'})$, such that the received signal is expressed as follows:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{w}. \tag{4.1}$$

On reception, a demodulator first computes a real-valued vector containing the n LLRs of the transmitted codeword $\tilde{\boldsymbol{c}}$ defined by:

$$\tilde{l}_i(y_{\lceil i/m \rceil}) = \log\left(\frac{P_{Y_{\lceil i/m \rceil}|\tilde{C}_i}(y_{\lceil i/m \rceil}|0)}{P_{Y_{\lceil i/m \rceil}|\tilde{C}_i}(y_{\lceil i/m \rceil}|1)}\right), \tag{4.2}$$

where $\boldsymbol{y} = (y_1, ..., y_{n'})$ is the complex-valued received signal. The resulting bit-LLRs are then de-interleaved back to the original bit order. This yields the vector $\boldsymbol{l} = (l_1, l_2, \ldots, l_n)$ that is then fed to the decoder to return an estimation $\hat{\boldsymbol{u}} = \boldsymbol{g}(\boldsymbol{l})$ of the originally transmitted message. As stated in Section 1.3.1, the decoding problem consists in designing a decoder $\boldsymbol{g}(\cdot)$ to minimize the BEP defined as in (1.27). The computation of the optimal decoding rule (see Section 1.3.1) requires the computation of the k posterior distributions $P_{U_i|\boldsymbol{L}}$ $\forall\, i \in [1 : k]$, entailing each one a marginalization over 2^{k-1} sequences $(u_1, ..., u_{i-1}, u_{i+1}, ..., u_k)$. This engenders exponential complexity and renders the bit-MAP decoder too complex for real-time implementation even for short codes, and simply intractable for larger codes. In this Chapter, we undertake the design of quasi-optimal and low complexity decoding rules $\boldsymbol{g}(\cdot)$ which are based on deep learning techniques.

4.1.2 BICM equivalent channel model

In order to derive optimal decoders for the BICM scenario under investigation, let us characterize the effective channel relating each of the transmitted codewords $\boldsymbol{c}$ to the received

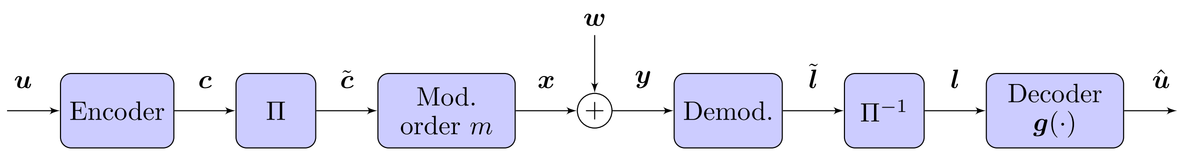


Figure 4.1: General system model for BICM.

LLR $\boldsymbol{l}$, namely the BICM equivalent channel model. Using results from the literature [Alv08; CTB98], the BICM channel depicted in Figure 4.2 can be characterized as a *memoryless* channel experienced equally by all of the coded bits. It consists of a mixture of the channels seen by each bit-position $s \in [1 : m]$ in the labeling of the constellation. More formally, the equivalent BICM channel is given in the following lemma.

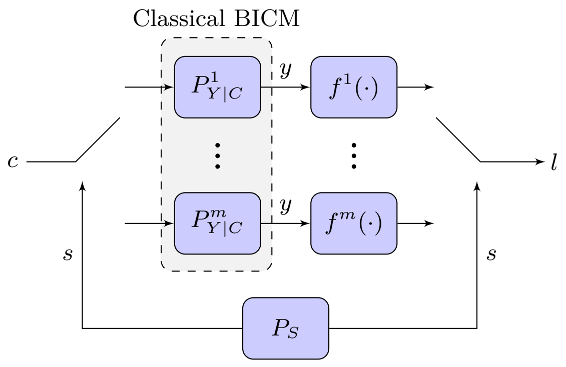


Figure 4.2: BICM channel model extended to bit-LLRs.

Lemma 4.1 (BICM channel model [Alv08; CTB98])**.** *An equivalent channel distribution of a BICM scenario writes for* $\boldsymbol{y} \in \mathbb{C}^{n'}$, $\boldsymbol{l} \in \mathbb{R}^n$, *and* $\boldsymbol{c} \in \{0,1\}^n$ *as:*

$$P_{\boldsymbol{L}|\boldsymbol{C}}(\boldsymbol{l}|\boldsymbol{c}) = \prod_{i=1}^{n} P_{L|C}(l_i|c_i), \tag{4.3}$$

where the distribution $P_{L|C}$ *is given by:*

$$P_{L|C}(l|c) = \frac{1}{m|\mathcal{X}_c^s|} \sum_{s=1}^{m} \sum_{x \in \mathcal{X}_c^s} P_{L_s|X}(l|x), \tag{4.4}$$

and $\mathcal{X}_c^s$ *is the set of symbols for which their* s*-th bit equals* c *(0 or 1). The distribution* $P_{L_s|X}$ *can be obtained from* $P_{Y|X}$, L_s *being a deterministic function of* Y.

Proof. The proofs can be found in [Alv08, Section 3.4] and follow from the fact that the equivalent channel relating $\boldsymbol{c}$ to $\boldsymbol{y}$ is memoryless, i.e.,

$$P_{\boldsymbol{Y}|\boldsymbol{C}}(\boldsymbol{y}|\boldsymbol{c}) = \prod_{i=1}^{n} P_{Y|C}(y_{\lfloor i/m \rfloor}|c_i), \tag{4.5}$$

where

$$P_{Y|C}(y|c) = \frac{1}{m|\mathcal{X}_c^s|} \sum_{s=1}^{m} \sum_{x \in \mathcal{X}_c^s} P_{Y|X}(y|x), \tag{4.6}$$

and $P_{Y|X}$ is the noisy channel probability distribution. □

In this work, we will not seek a closed-form expression of the BICM equivalent channel distribution $P_{L|C}$. However, the equivalent channel model will prove useful to analyze the optimality of our proposed decoder in Section 4.3.

4.2 Model-free decoding for BICM

In the following, we propose a model-free decoder for higher-order modulations under a BICM scenario. To this end, we start by recalling the principle of model-free decoding, then present the structure of the proposed decoder and compare it to the existing state of the art.

4.2.1 Model-free decoding for BPSK and QPSK

The model-free decoder introduced in [BCK18a] consists of two main ideas, namely *estimating bitflips* in the received noisy codewords and identifying *sufficient statistics* to do so.

To this end, a pre-processing stage is added before the decoder, where the received BPSK signal $\boldsymbol{y}$ is divided into its absolute value $|\boldsymbol{y}|$ and its syndrome $\boldsymbol{s}$, defined as:

$$\boldsymbol{s} = H\boldsymbol{y}^b, \tag{4.7}$$

where H denotes the parity-check matrix of the code, $\boldsymbol{y}^b$ represents the hard decision associated with the received signal $\boldsymbol{y}$. Observe that $\boldsymbol{y}^b = \boldsymbol{c} \oplus \boldsymbol{e}^b$ where $\boldsymbol{c}$ indicates the transmitted binary codeword and $\boldsymbol{e}^b$ is a vector containing ones in the flipped positions and zeros everywhere else —also referred to as the bitflip pattern. Authors in [BCK18a] proved that the two elements $(H\boldsymbol{y}^b, |\boldsymbol{y}|)$ are sufficient statistics for the optimal estimation of the bitflip pattern $\boldsymbol{e}^b$ and are, by definition, independent of the transmitted codeword $\boldsymbol{c}$ when employing a BPSK modulation. A schematic representation of this decoding paradigm is shown in Figure 4.3. This implies that (i) optimal decoding can be achieved with this framework, and (ii) training can be carried out using noisy observations of a single codeword.

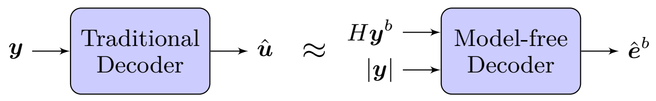


Figure 4.3: Equivalence between traditional decoding (left) and the model-free approach (right) as presented in [BCK18a] for a BPSK modulation scheme. The estimated bitflip pattern $\hat{\boldsymbol{e}}^b$ is used to correct the hard-decision vector $\boldsymbol{y}^b$.

Although originally designed for a BPSK modulation, the extension of the results of [BCK18a] to a QPSK scenario can be done readily by decomposing $\boldsymbol{y}$ into its real and imaginary parts $(\mathrm{Re}(\boldsymbol{y}), \mathrm{Im}(\boldsymbol{y}))$, and then extracting the absolute value and syndrome associated with the serial concatenation of both the real and imaginary components.

4.2.2 Model-free decoding for higher-order BICM

In this section, we present our proposed model-free decoder for higher-order modulations, which is an extension of the decoding framework of Chapter 3. To this end, we suggest using the decoding architecture of Figure 4.4. Let $\mathcal{C}$ be an (n, k) linear block code defined by its generator and parity-check matrices G and H, respectively, and let $\boldsymbol{l}^b$ denote the hard decisions associated with the deinterleaved LLR vector $\boldsymbol{l}$. The proposed decoder maps each received LLR sequence $\boldsymbol{l}$ into an estimate of the original message $\hat{\boldsymbol{u}}$ as follows:

(i) The deinterleaved LLR sequence $\boldsymbol{l}$ delivered by the demodulator undergoes a pre-processing stage that extracts the syndrome of its associated hard decisions (i.e., $H\boldsymbol{l}^b$), and the absolute value of each component (i.e., $|\boldsymbol{l}|$), also referred to as *reliabilities*.

(ii) A primary estimate of the message $\tilde{\boldsymbol{u}}$ is then obtained through the pseudo-inverse of the LLR hard decisions $\boldsymbol{l}^b$ as follows:

$$\tilde{\boldsymbol{u}} \triangleq \mathrm{pinv}(\boldsymbol{l}^b), \tag{4.8}$$

where $\mathrm{pinv}(\cdot)$ corresponds to a pseudo-inverse of the code $\mathcal{C}$ such that, for every valid codeword $\boldsymbol{c} = G^T\boldsymbol{u}$, it holds that $\mathrm{pinv}(\boldsymbol{c}) = \boldsymbol{u}$. Observe that if a realization of the LLR vector has suffered a sign change, i.e., $\boldsymbol{l}^b \neq \boldsymbol{c}$, then the auxiliary message sequence $\tilde{\boldsymbol{u}}$ might also contain errors. Observe also that, in the special case of a systematic code where the first k bits in the codeword correspond to the information bits, a simple pseudo-inverse corresponds to extracting the systematic bits, i.e., $\mathrm{pinv}(\boldsymbol{c}) = A\boldsymbol{c}$ where:

$$A = \left[\, I_k \,|\, \mathbf{0}_{k,n-k} \,\right]. \tag{4.9}$$

(iii) The two inputs $(|\boldsymbol{l}|, H\boldsymbol{l}^b)$ are fed to the message bitflip estimator, which outputs a vector $\hat{\boldsymbol{e}}_{\boldsymbol{u}}$ with positive values in positions where a bitflip occurred in $\tilde{\boldsymbol{u}}$ and negative values otherwise. The $\mathrm{bin}(\cdot)$ operation converts the real values into binary as follows:

$$\hat{\boldsymbol{e}}^b_{\boldsymbol{u}} = \mathrm{bin}(\hat{\boldsymbol{e}}_{\boldsymbol{u}}) \triangleq \mathbb{1}(\hat{e}_{u,i} > 0)_{1 \leq i \leq k}. \tag{4.10}$$

(iv) Finally, $\hat{\boldsymbol{e}}^b_{\boldsymbol{u}}$ is used to correct $\tilde{\boldsymbol{u}}$ of (4.8) and obtain the estimated message $\hat{\boldsymbol{u}} = \tilde{\boldsymbol{u}} \oplus \hat{\boldsymbol{e}}^b_{\boldsymbol{u}}$.

This decoding layout is also referred to as SBND, as in Chapter 3. This is because even if bit-interleaving is added to regain the necessary symmetry properties, the decoder's structure remains essentially the same as before, and this version can be straightforwardly applied to the BPSK scenario. It is worth mentioning that the design and training of the message bitflip estimator will be tackled in the upcoming sections.

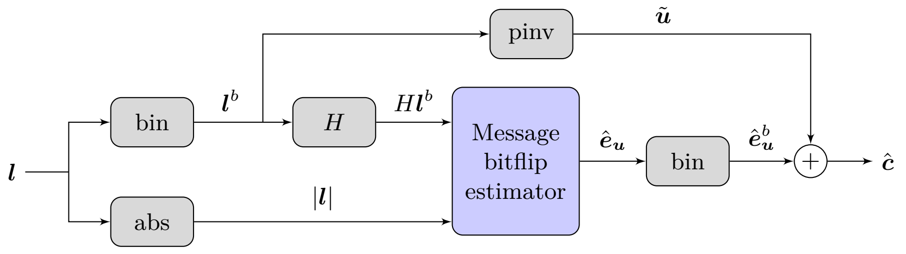


Figure 4.4: Proposed architecture for the channel decoder in a BICM scenario. No assumption has been made so far regarding the implementation of the bitflip estimator.

4.2.3 Comparison with related works

The hereinbefore proposed decoder, which extends the framework proposed in Chapter 3 to admit higher-order modulations, improves on that in previous works [BCK18a; CW22a; CW23; KP20; Par+23] in three different aspects.

First, it exploits the previous improvement suggested in Chapter 3 seeking to estimate the bitflips induced by the channel on the message bits $\boldsymbol{u}$ instead of on the whole codeword $\boldsymbol{c}$. The improvements entailed by this feature are two-fold. On the one side, since the objective function to be optimized is the BEP defined as per (1.27), restricting the loss function to the message $\boldsymbol{u}$ instead of the whole codeword $\boldsymbol{c}$ allows to disregard error events on the $n-k$ parity bits and reduces the decision space dimension from n to k. On the other side, the present decoder structure discards error events in which the obtained *denoised* codeword is not a valid codeword, i.e., does not belong to the code.

Second, as in Chapter 3, the proposed decoder is directly applicable to non-systematic codes, without the need to apply a linear transformation to obtain the estimated message $\hat{\boldsymbol{u}}$ from the estimated codeword $\hat{\boldsymbol{c}}$. The pseudo-inverse introduced in the proposed decoder is such that the neural network estimates the bitflips present in an artificial variable that has already addressed the codeword-to-message operation.

Third, the proposed decoder generalizes the structures of previous works [BCK18a; CW22a; CW23; KP20; Par+23] and Chapter 3 from a BPSK-specific decoding system to a decoder suited for arbitrary higher-order modulations by considering the LLR vector $\boldsymbol{l}$ instead of the received noisy symbols $\boldsymbol{y}$. This is the aspect studied in this Chapter. As we show in Section 4.3, the proposed decoder can achieve optimal decoding for the BICM scenario in the sense that it can approximate the MAP decoder. However, the theoretical analysis of the proposed decoder is more involved than in the case of a BPSK modulation scheme.

4.3 On the optimality of SBND for BICM

In the following, we prove that the suggested decoder can allow us to approximate the MAP criterion for the specific case of BICM. Besides, we discuss the design of the training dataset and derive a training strategy for the message bitflip estimator of the decoder. It is worth mentioning that, for space limitations, we perform the analysis only for the 8-PSK and 16-QAM constellations depicted in Figure 4.5. However, the analysis can be carried out similarly for arbitrary M-PSK and M-QAM with Gray-labelling.

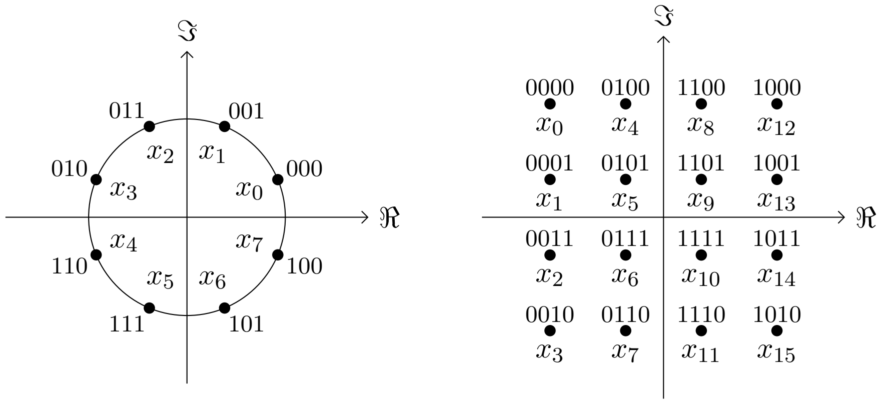


Figure 4.5: Constellations for 8-PSK (left) and 16-QAM (right) modulation schemes.

4.3.1 LLR expressions for 8-PSK and 16-QAM

Since the proposed decoder is based on the received LLR, let us start by giving approximate closed-form expressions of the received LLRs as a function of the received signal $\boldsymbol{y}$, for every bit position of the constellation. The closed-form expressions hereafter are based on the *approximate* LLRs that are obtained as follows. Let $s \in [1:m]$ be the index of a bit position in the constellation labeling and let y be a received noisy symbol. We can write that

$$
\begin{aligned}
l_s(y) &= \log\Big(\sum_{x\in\mathcal{X}_0^s} P_{Y|X}(y|x)\Big) - \log\Big(\sum_{x\in\mathcal{X}_1^s} P_{Y|X}(y|x))\Big) && (4.11)\\
&\approx \log\left(\max_{x\in\mathcal{X}_0^s} P_{Y|X}(y|x)\right) - \log\left(\max_{x\in\mathcal{X}_1^s} P_{Y|X}(y|x)\right) && (4.12)\\
&= \frac{1}{\sigma^2}(\min_{x\in\mathcal{X}_1^s} ||y-x||^2 - \min_{x\in\mathcal{X}_0^s} ||y-x||^2) && (4.13)\\
&= \gamma(\min_{x\in\mathcal{X}_1^s} ||y-x||^2 - \min_{x\in\mathcal{X}_0^s} ||y-x||^2), && (4.14)
\end{aligned}
$$

where $\mathcal{X}_0^s$ (resp. $\mathcal{X}_1^s$) is the set of symbols of the constellation for which the s-th bit is equal to 0 (resp. to 1), and where $\gamma \triangleq 1/\sigma^2$. Note that these approximate LLR expressions are tight for moderate to high SNRs.

Lemma 4.2 (LLRs of an 8-PSK with Gray labeling)**.** *Under the Gray mapping of Figure 4.5, the approximate LLRs for each of the three bit-positions of an 8-PSK are given by:*

$$l_1(y) = \begin{cases} 4\gamma\alpha \mathrm{Im}(y) & \textit{if } |\mathrm{Im}(y)| \leq |\mathrm{Re}(y)| \\ 2\sqrt{2}\gamma \ \mathrm{sign}(\mathrm{Im}(y))\,(\beta|\mathrm{Im}(y)| - \alpha|\mathrm{Re}(y)|) & \textit{else,} \end{cases}$$

$$l_2(y) = \begin{cases} 4\gamma\alpha \mathrm{Re}(y) & \textit{if } |\mathrm{Re}(y)| \leq |\mathrm{Im}(y)| \\ 2\sqrt{2}\gamma \ \mathrm{sign}(\mathrm{Re}(y))\,(\beta|\mathrm{Re}(y)| - \alpha|\mathrm{Im}(y)|) & \textit{else,} \end{cases}$$

$$l_3(y) = 2\gamma(\beta - \alpha)(|\mathrm{Im}(y)| - |\mathrm{Re}(y)|),$$

where $\alpha \triangleq \sin(\pi/8)$ *and* $\beta \triangleq \cos(\pi/8)$*.*

Proof. The proof is given in Appendix C.1.1. □

Lemma 4.3 (LLRs of a 16-QAM with Gray labeling)**.** *As for the 16-QAM, under the Gray labeling of Figure 4.5, the approximate LLRs for each of the four bit-positions are given by:*

$$l_1(y) = \begin{cases} -2\gamma d \mathrm{Re}(y) & \textit{if } |\mathrm{Re}(y)| \leq d \\ -2\gamma d \ \mathrm{sign}(\mathrm{Re}(y))\,(2|\mathrm{Re}(y)| - d) & \textit{else,} \end{cases} \tag{4.15}$$

$$l_3(y) = \begin{cases} 2\gamma d \mathrm{Im}(y) & \textit{if } |\mathrm{Im}(y)| \leq d \\ 2\gamma d \ \mathrm{sign}(\mathrm{Im}(y))\,(2|\mathrm{Im}(y)| - d) & \textit{else,} \end{cases} \tag{4.16}$$

$$l_2(y) = 2\gamma d(|\mathrm{Re}(y)| - d), \tag{4.17}$$

$$l_4(y) = 2\gamma d(|\mathrm{Im}(y)| - d), \tag{4.18}$$

where d is the minimum distance of the constellation and is given by $d = \sqrt{2/5}$.

Proof. The proof is given in Appendix C.1.2. □

The expressions herebefore stated will allow us to characterize each of the channels relating the reliabilities $|\boldsymbol{l}|$ and hard LLRs $\boldsymbol{l}^b$ to the transmitted codewords $\boldsymbol{c}$, which is crucial to analyze the optimality of the decoder and justify the design of the training set of codewords.

4.3.2 Optimality for BICM scenario

In the following, we seek to show that for the BICM scenario, provided that the message bitflip estimator of Figure 4.4 is properly designed, the proposed decoder can achieve performances that are close to the MAP decoder.

In order to prove that the proposed decoder is optimal for the BICM scenario, we need to characterize the channel distribution $P_{\boldsymbol{L}^b|\boldsymbol{C}}$. To this end, let us consider the 16-QAM and 8-PSK constellations with Gray labeling shown in Figure 4.5. The following result allows us to characterize the channel $P_{\boldsymbol{L}^b|\boldsymbol{C}}$.

Theorem 4.1 (Bit-LLRs binary channel model)**.** *For all $\boldsymbol{l}^b, \boldsymbol{c} \in \{0,1\}^n$, the following holds:*

$$P_{\boldsymbol{L}^b|\boldsymbol{C}}(\boldsymbol{l}^b|\boldsymbol{c}) = \prod_{i=1}^{n} P_{L^b|C}(l_i^b|c_i). \tag{4.19}$$

Besides, for the 8-PSK and 16-QAM under Gray labeling, the channel $P_{L^b|C}$ can be approximated by:

$$\boldsymbol{L}^b = \boldsymbol{C} \oplus \boldsymbol{E}^b \quad s.t. \quad \boldsymbol{E}^b \overset{i.i.d}{\sim} Bern(q), \tag{4.20}$$

where $\boldsymbol{E}^b$ is independent of $\boldsymbol{C}$ and $q \triangleq \frac{1}{m}\sum_{s=1}^{m} P_{L_s^b|C}(1|0)$.

Proof. The detailed proof is relegated to Appendix C.2, yet, an outline of the proof is given below. First, by Lemma 4.1, the channel $P_{\boldsymbol{L}|\boldsymbol{C}}$ is memoryless, hence, the channel $P_{\boldsymbol{L}^b|\boldsymbol{C}}$ is also memoryless. Next, given the LLR expressions in Lemmas 4.2 and 4.3, then $P_{L^b|C}(1|0) = P_{L^b|C}(0|1)$ (for all SNR for the 8-PSK, and for intermediate to high SNR for 16-QAM). This allows us to state that the binary channel relating C to L^b can be approximated with a Binary Symmetric Channel (BSC) and, therefore, admits an equivalent additive binary noise model as in (4.20). □

Note that, unlike the involved proof needed for higher-order modulations, this Lemma can be proved easily for BPSK and extended to QPSK modulations recovering the results of Chapter 3.

Having characterized the binary channel relating $\boldsymbol{C}$ to $\boldsymbol{L}^b$ as a memoryless BSC, we state the main result that proves the optimality of the proposed decoder for the BICM case.

Theorem 4.2 (Sufficient statistics)**.** *Considering the BICM scenario, and the result of Theorem 4.1, then for all $\boldsymbol{e}_{\boldsymbol{u}}^b \in \{0,1\}^k$ and $\boldsymbol{l} \in \mathbb{R}^n$, we have that:*

$$P_{\boldsymbol{E}_{\boldsymbol{u}}^b|\boldsymbol{L}}(\boldsymbol{e}_{\boldsymbol{u}}^b|\boldsymbol{l}) = P_{\boldsymbol{E}_{\boldsymbol{u}}^b|\,|\boldsymbol{L}|,H\boldsymbol{L}^b}(\boldsymbol{e}_{\boldsymbol{u}}^b|\,|\boldsymbol{l}|, H\boldsymbol{l}^b). \tag{4.21}$$

Hence, it follows that:

$$P_{E_{u,i}^b|\boldsymbol{L}}(e_{u,i}^b|\boldsymbol{l}) = P_{E_{u,i}^b|\,|\boldsymbol{L}|,H\boldsymbol{L}^b}(e_{u,i}^b|\,|\boldsymbol{l}|, H\boldsymbol{l}^b). \tag{4.22}$$

Proof. The proof is relegated to Appendix C.3. □

This result allows us to state that, in the case of BICM, the posterior distribution of the message bitflips $\boldsymbol{e}_{\boldsymbol{u}}^b$ given the received LLR $\boldsymbol{l}$ is indeed equal to the posterior distribution of the bitflips $\boldsymbol{e}_{\boldsymbol{u}}^b$ given only the reliabilities and syndrome $(|\boldsymbol{l}|, H\boldsymbol{l}^b)$. As such, the quantities $(|\boldsymbol{l}|, H\boldsymbol{l}^b)$ offer sufficient statistics for estimating $\boldsymbol{e}_{\boldsymbol{u}}^b$. Thus, provided that the message bitflip estimator yields a good approximation of the posterior distribution $P_{\boldsymbol{E}_{\boldsymbol{u}}^b|\,|\boldsymbol{L}|,H\boldsymbol{L}^b}$, the proposed decoder will be close to the optimal.

4.3.3 Training set design

The performance of the proposed decoder depends on the capacity of the message bitflip estimator in Figure 4.4 to yield a good approximation of the posterior distribution $P_{\boldsymbol{E}_{\boldsymbol{u}}^b||\boldsymbol{L}|,\boldsymbol{H}\boldsymbol{L}^b}$, or rather the argmax of said posterior. In this work, we choose to implement the message bitflip estimator using neural networks since they are well-known and powerful universal approximators [HSW89]. The training loss is set to be the *binary cross-entropy* which will further ensure that the soft output of the neural network converges to the desired posterior distribution, provided that the training dataset is properly designed to prevent overfitting.

Let us now analyze the design of the training dataset. A key implication of the result of Theorem 4.2 when applied to the case of low-order constellations, namely BPSK and QPSK, is that the quantities $|\boldsymbol{l}|$ and $\boldsymbol{H}\boldsymbol{l}^b$ are invariant to the transmitted codewords $\boldsymbol{c}$, reducing thus the training set to only one codeword $\boldsymbol{c}$ (for instance the all-zero codeword) as shown in Chapter 3. This feature is key to the scaling capability of syndrome-based decoders for BPSK and QPSK since it alleviates the necessity to train over the whole set of all 2^k possible codewords $\boldsymbol{c}$.

For higher-order modulations such as the 8-PSK and 16-QAM, while the syndrome $\boldsymbol{H}\boldsymbol{l}^b$ is independent of the transmitted codeword $\boldsymbol{c}$ in a BICM scenario, the reliabilities $|\boldsymbol{l}|$ are not —they depend on the transmitted symbols $\boldsymbol{x}$. However, some invariances can be identified, which will allow for the reduction of the training dataset to fewer codewords than the whole codebook. The following result describes the invariances of these reliabilities.

Lemma 4.4 (Invariances of the reliabilities $|\boldsymbol{l}|$)**.** *Let $y \in \mathbb{C}$ be a noisy received symbol such that $y = x + w$ where x is the transmitted symbol in a given constellation and w is a realization of a $\mathcal{CN}(0, \sigma^2 I_2)$ AWGN. Given the LLR formulae in Lemmas 4.2 and 4.3, the following holds.*

1) *For the 8-PSK of Figure 4.5 and all $s \in [1:m]$, there exists a realization of the AWGN w' such that $P_W(w) = P_W(w')$ and*

$$|l_s(y)| = |l_s(x_0 + w')| \text{ or } |l_s(y)| = |l_s(x_1 + w')|, \tag{4.23}$$

where x_0 and x_1 are as depicted in Figure 4.5.

2) *Similarly, for the 16-QAM constellation of Figure 4.5 and all $s \in [1:m]$, there exists a realization of the AWGN w' such that $P_W(w) = P_W(w')$ and*

$$\begin{aligned} &|l_s(y)| = |l_s(x_0 + w')| \\ \text{or } &|l_s(y)| = |l_s(x_1 + w')| \\ \text{or } &|l_s(y)| = |l_s(x_4 + w')|, \end{aligned} \tag{4.24}$$

where x_0, x_1, x_4 are as depicted in Figure 4.5.

Proof. The proof is relegated to Appendix C.4. □

The main implication of this result is that, although the reliabilities $|l_s(y)|$ depend on the transmitted symbol x, they have inherent symmetries that allow all possible values of $|l_s(y)|$ to be readily observed from only a reduced set of *representative symbols* (for instance x_0 and x_1 for the 8-PSK) with the same probability.

In the absence of the linear block code, i.e. $k = n$, the present result allows to reduce the set of all possible training symbol sequences $\boldsymbol{x}$ from $8^{n'}$ to $2^{n'}$ for the 8-PSK, and from $16^{n'}$ to $3^{n'}$ for the 16-QAM, which reduces greatly the number of symbol sequences to be seen during training. However, the gain in the case of a (n, k) linear block code is more challenging to assess analytically due to the intricate dependence on the code structure as not all symbol sequences $\boldsymbol{x}$ are possible.

Yet, in light of this result, we can readily infer that the choice of the all-zero codeword training strategy would allow us to see only one statistic of the reliabilities $|l_s|$ throughout the training (that of x_0 for the 8-PSK for instance), which would lead eventually to overfitting. A first improvement of this strategy would consist in selecting a unique training codeword constrained to a Hamming weight as close as possible to $n/2$. This would allow us to see, at each bit position i, more possible statistics of the reliabilities $|l_s|$. In the following numerical simulations, we will employ a randomly generated batch of codewords at each training epoch, which ensures a maximum variety of reliability combinations for each bit position. Nonetheless, with respect to the code lengths we employ during the simulations ($n \geq 63$), the number of codewords transmitted during training is but a negligible fraction of the total number of possible codewords.

4.4 Experiments

Code	Rate	Mod.	Training E_b/N_0	RNN	ECCT	r-ECCT
BCH $(63, 57)$	0.90	8-PSK	6dB	$\alpha = 5$, $d_l = 5$, b.s. $= 2^{12}$	$d_e = 128$, $N = 10$, b.s. $= 2^8$	$d_e = 128$, $N = 10$, b.s. $= 2^9$
BCH $(63, 51)$	0.81					
BCH $(63, 45)$	0.71		5dB			
BCH $(63, 39)$	0.62					
Polar $(64, 48)$	0.75	16-QAM	6dB	$\alpha = 5$, $d_l = 5$, b.s. $= 2^{12}$	$d_e = 128$, $N = 10$, b.s. $= 2^8$	$d_e = 128$, $N = 10$, b.s. $= 2^9$
Polar $(64, 32)$	0.50		5dB			
Polar $(64, 22)$	0.34					
Polar $(128, 96)$	0.75		6dB			
Polar $(128, 64)$	0.50		5dB			

Table 4.1: Summary of the hyperparameters used in each simulated scenario, along with the approximate number of weights that configure each NN architecture.

In this section, we present the decoding performance of our proposed decoder from Section

4.2.2 with two different families of codes, namely BCH and Polar codes, each with different sizes and code rates. To this end, we implement the three main architectures from Sections 3.1.5 and 3.2.2 of Chapter 3 (namely, RNN, ECCT and r-ECCT) using TensorFlow [Mar+15] along with the Keras API [Cho+15]. We resort to the *binary cross-entropy* as a training loss function defined for $\boldsymbol{e} = \{e_i\}_{1\leq i\leq k}$ and $\hat{\boldsymbol{e}} = \{\hat{e}_i\}_{1\leq i\leq k}$ as follows:

$$\mathcal{L}_{\text{BCE}}(\boldsymbol{e}, \hat{\boldsymbol{e}}) = \sum_{i=1}^{k}\Big(e_i\log(\hat{e}_i) + (1-e_i)\log(1-\hat{e}_i)\Big), \tag{4.25}$$

where $\boldsymbol{e}$ is conventionally the true value (or *ground truth*) and $\hat{\boldsymbol{e}}$ denotes the NN-estimated value. As for the input vector of the NN, i.e., $(|\boldsymbol{l}|, \boldsymbol{H}\boldsymbol{l}^b)$, we normalize the reliabilities $|\boldsymbol{l}|$ by multiplying them by a factor of σ^2 to render the input range less dependent on the SNR. We observed that this normalization operation yields a better generalization capability of the message bitflip estimator for a wide range of SNRs, and does not carry any further knowledge assumption as σ^2 is already considered known to the demodulator. Regarding the output of the NN, we fix the last dense layer of every architecture to be linear to *mimic* LLR values. As such, a sigmoid function is applied to the estimators' outputs before computing the binary cross-entropy loss. Another possibility would be to select the sigmoid as the final activation function and use a 1/2-threshold to convert to the binary domain. For reproducibility, the architecture and training hyperparameters employed are summarized in Table 4.1, and a complexity summary is presented in Table 4.2.

Figures 4.6 and 4.7 display the BER performances of several BCH codes using an 8-PSK modulation scheme. The SBND approach of section 4.2 is implemented using the three studied architectures. As a lower bound on the BER of the proposed decoder, the ML bound of Section 1.4.3 is derived as in [TV11], using an OSD [FL95] of sufficiently high order (2 or 3, depending on the code), through the following process. Whenever a decoding failure occurs for the OSD, we assess whether the decoded codeword is more likely than the transmitted one. Should this be the case, it implies that even an ML decoder would have produced a decoding error. This algorithm yields a lower bound on the error probability, yet in our scenarios, it closely aligns with the performance of the OSD when a sufficiently high order is selected.

For the BCH $(63, 57)$ and $(63, 51)$, all architectures displayed optimal and close-to-optimal decoding performances, respectively. Across all cases, particularly for the most challenging codes, such as BCH $(63, 45)$ and $(63, 39)$, the RNN-based architecture exhibited the best decoding performance, closely approaching that of the ML decoder for the considered codes. However, as discussed in Section 3.2.3, this architecture is also the most complex since its number of parameters increases with the input size $2n-k$. The hyperparameters of the ECCT were fixed across all codes to maintain a constant number of weights, with a batch size set to 2^8. As for the proposed architecture, the r-ECCT outperformes the ECCT when applied to lower rate codes, while employing only 10% of the total number of weights. This improvement can be partially attributed to the smaller architecture, which allows for an increase in the training batch size to 2^9, enabling a more precise gradient calculation.

Figure 4.8 shows the BER performances of each architecture for three Polar codes with

Code	W_{RNN}	W_{ECCT}	$W_{r\text{-}ECCT}$
BCH $(63, 57)$	3.3M	2M	200k
BCH $(63, 51)$	3.9M		
BCH $(63, 45)$	4.5M		
BCH $(63, 39)$	5.2M		
Polar$(64, 48)$	4.4M	2M	200k
Polar $(64, 32)$	6.4M		
Polar $(64, 22)$	7.8M		
Polar $(128, 96)$	17.8M		
Polar $(128, 64)$	25.5M		

Table 4.2: Summary of the hyperparameters used in each simulated scenario, along with the approximate number of weights that configure each NN architecture.

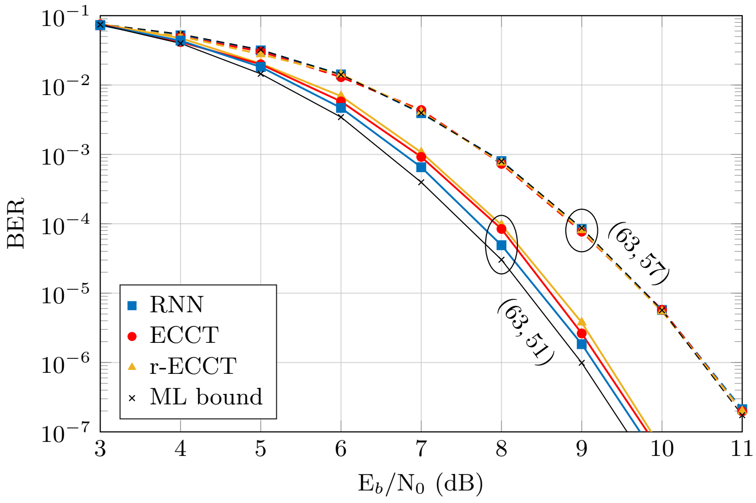
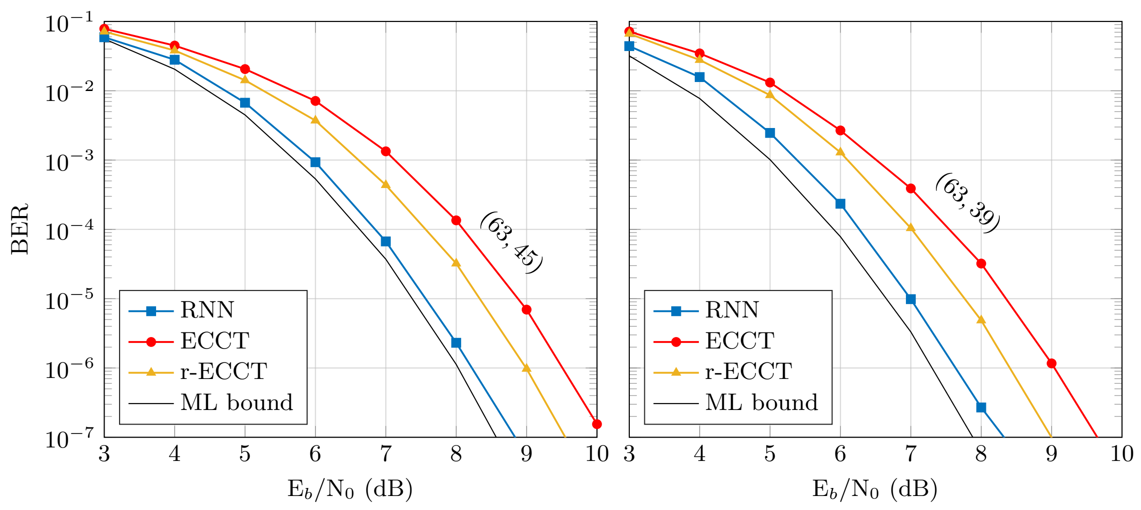


Figure 4.6: BER comparison for two BCH codes of sizes $(63, 57)$ (dashed lines) and $(63, 51)$ (solid lines) and an 8-PSK modulation scheme using the SBND, employing the three considered architectures. The ML curve was computed using an OSD of order 2.



(a) BCH $(63, 45)$.

(b) BCH $(63, 39)$.

Figure 4.7: BER comparison for two BCH codes using an 8-PSK modulation scheme, employing the three considered architectures. The ML decoder curves were computed using an OSD of order 3.

$n = 64$ and $k = \{48, 32, 22\}$, employing a 16-QAM modulation scheme. In this case, even though the lower-rate codes are harder for the network to learn, the performances of the NN decoders are considerably closer to those of the ML decoder than those of the BCH. Due to its high-density nature, the latter code is usually considered *harder* for deep learning-based solutions [BCK18a; CW22a; NBB16; NW21]. Hence, it can be observed that the SBND approach applied to Polar codes displays error rates closer to the ML decoder. For this reason, Figure 4.9 exhibits the BER performances of two larger Polar codes, of sizes $(128, 96)$ and $(128, 64)$. As in the previous examples, for the same value of n, higher-rate codes are more easily learnable by the SBND, with the three architectures producing very similar error curves that approach the ML decoder for the Polar $(128, 96)$. With $k = 64$, however, a larger gap appears between the optimal decoder and the SBND.

Last but not least, we analyze in Figure 4.10 the effect of the training dataset design —in the light of the analysis of Section 4.3.3— by comparing three training dataset design strategies: (i) the common *all-zero codeword* training strategy of [BCK18a] and Chapter 3; (ii) a codeword with Hamming weight equal to $n/2$ termed *equilibrated codeword*; and (iii) random codeword generation for every sample. It can be seen that, as expected, the all-zero codeword is detrimental to the learning capability of the NN, while the equilibrated codeword allows it to achieve satisfactory performances. However, the better strategy remains the random codeword generation, and as such, was adopted through the numerical analysis herebefore.

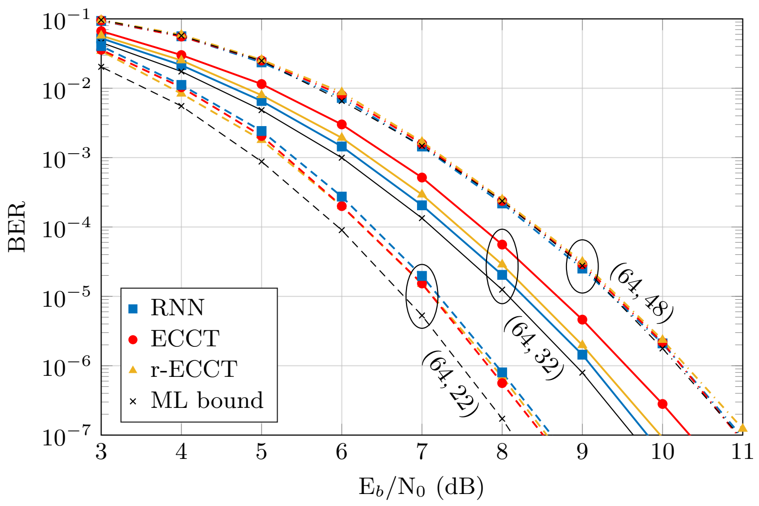


Figure 4.8: BER comparison for three Polar codes of sizes $(64, 48)$ (dash-dotted lines), $(64, 32)$ (solid lines), and $(64, 22)$ (dashed lines), and a 16-QAM modulation scheme, employing the three considered architectures. The MLB curve was computed using an OSD of order 3.

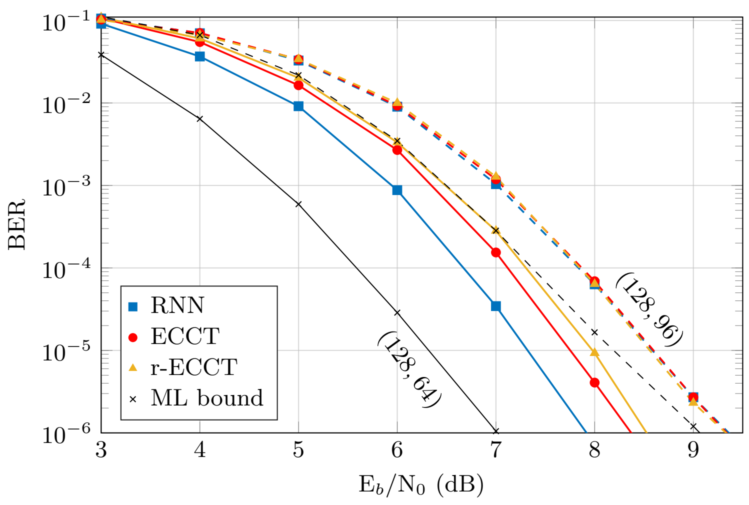


Figure 4.9: BER comparison for two Polar codes of sizes $(128, 96)$ (dashed lines) and $(128, 64)$ (solid lines), and a 16-QAM modulation scheme, employing the three considered architectures. The MLB curve was computed using an OSD of sufficiently high order.

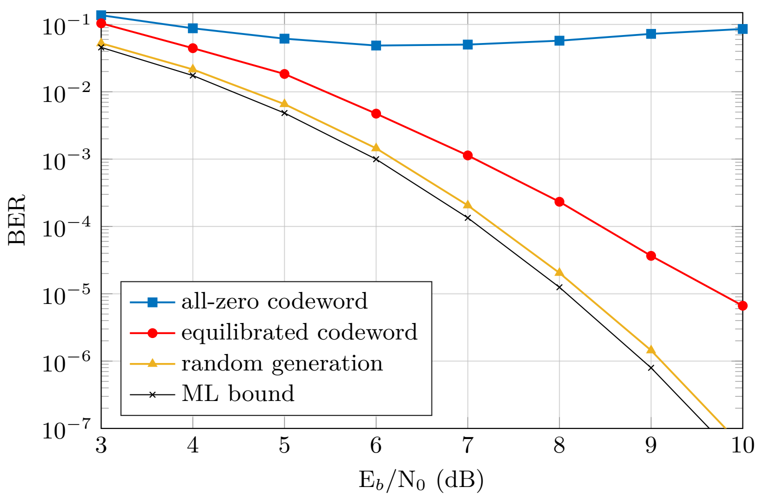


Figure 4.10: BER comparison for a Polar code of size $(64, 32)$ using a 16-QAM modulation and an RNN architecture. For training, the three considered types of codeword generation were employed.

4.5 Discussion: SBND for a generic CM scenario

Throughout the present Chapter, the BICM scenario has been extensively investigated, both analytically and numerically. However, under a generic CM scenario, the channel relating the codeword $\boldsymbol{c}$ to the received LLRs $\boldsymbol{l}$ is no longer fully memoryless in that: (i) each of the coded bits c_i experiences a different channel statistic that depends on its position $s \in [1 : m]$ in the labeling of the constellation; and (ii) the channels experienced by each of the bits $\boldsymbol{c}_i \triangleq (c_{\lceil i/m \rceil}, \cdots, c_{\lceil i/m \rceil + m - 1})$ corresponding to the same symbol $x_{\lceil i/m \rceil}$ are no longer parallel and independent as in Figure 4.2. As such, (4.3) writes as

$$P_{\boldsymbol{L}|\boldsymbol{C}}(\boldsymbol{l}|\boldsymbol{c}) = \prod_{i=1}^{n'} P_{L_1,\ldots,L_m|C_1,\ldots,C_m}(\boldsymbol{l}_i|\boldsymbol{c}_i), \tag{4.26}$$

where $\boldsymbol{l}_i \triangleq (l_{\lceil i/m \rceil}, \cdots, l_{\lceil i/m \rceil + m - 1})$. Hence, the results of Theorems 4.1 and 4.2 do not hold anymore, rendering the proposed decoder suboptimal with respect to the MAP rule in (1.36). This feature, however, is frequent in other decoding algorithms that rely uniquely on the reliabilities and syndrome of the received LLRs values to produce an estimate of the message bits, such as the LLR-based OSD [FL95], or an estimate of the message bitflips such as ORB-GRAND [DAM22].

Figure 4.11 shows the performance comparison of the proposed decoder for a BICM and generic CM scenarios under an 8-PSK modulation scheme. Therein, we can observe that final performances are practically equivalent between both settings. Although we cannot prove analytically that the proposed decoder approaches the MAP decoder for a generic CM scenario, as we were able to do with the mathematical model of BICM, it still allows for competitive performances with existing decoders (e.g., OSD) that consider the same inputs (reliabilities and syndrome).

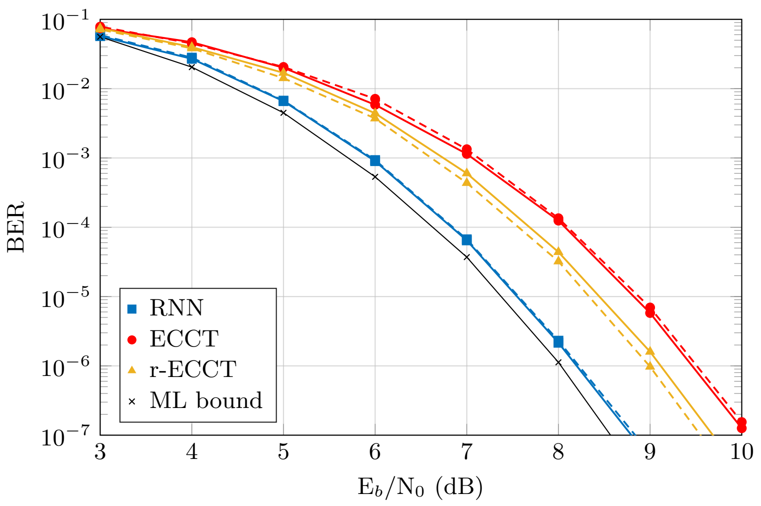


Figure 4.11: BER comparison for a BCH code of size $(63, 45)$ and an 8-PSK modulation scheme, employing the three considered architectures. Solid lines denote the BICM scenario, whereas dashed lines represent the generic, non-interleaved CM layout.

Conclusion and future work

Synthesis of the work

In this Ph.D. thesis, we have explored machine learning-based solutions for channel decoding, which constitutes a critical area in next-generation communication systems, such as Machine-to-Machine (M2M) type communications. In this scenario, we seek ultra-reliable low-latency communications (URLLC), which call for shorter block lengths than those used in other applications. Classical decoding methods often rely on asymptotic behavior and fail to provide optimal solutions for short block lengths, and optimal decoders are usually very complex. At the same time, data-driven solutions rapidly meet the *scalability* problem, where codes with block lengths larger than a few tens of bits involve intractably large neural-based architectures and training. This work undertakes this problem, proposing solutions to take a step further toward realistic implementations of low-complexity and competitive machine learning-based decoders.

After providing the reader with a brief introduction to channel coding and linear modulations in Chapter 1, we presented Support Vector Machines (SVM) in Chapter 2 and introduced a novel bit-wise approach, which greatly reduces the decoder's complexity compared to previous SVM-based solutions. We further developed our study to prove the equivalence between the proposed decoder and the Maximum Likelihood (ML) decoder under an Additive White Gaussian Noise (AWGN) channel. Simulations over short BCH and Polar codes verify our findings and the importance of properly selecting the optimization hyperparameters.

In Chapter 3, we presented a novel *message-oriented* decoding framework that builds upon previous works and employs Neural Networks (NN) to estimate the positions in the message that suffered bitflips. This approach gained popularity in the past few years due to its scalability properties, as the decoder can be fully trained on a single codeword (for example, the all-zero codeword). Our approach's novelty lies in its focus on the information bits rather than the entire codeword, which translates into a significant performance improvement and direct applicability to non-systematic codes. Next, we confronted the network size problem by presenting a low-complexity version of an existing transformer-based architecture, which maintains performances with only a fraction of the number of weights. Finally, we addressed a previously unexplored area by conducting a study using information theory-based metrics to analyze the impact of the parity-check matrix on the decoder's training and performance. We proposed an algorithm that adapts the parity-check matrix accordingly, which results in significant performance improvement and smoother training, without employing any additional computational power in decoding. All the results are supported by specific simulations addressing each contribution separately.

In Chapter 4, we adapted the decoding framework of the previous Chapter —which is specific to Binary Phase-Shift Keying (BPSK)— to make it applicable to higher-order modulations, such as Phase-Shift Keying (PSK) and Quadrature Amplitude Modulation (QAM). For this purpose, we resorted to Bit-Interleaved Coded Modulations (BICM) to regain the necessary symmetry properties to analytically prove the decoder's optimality. The same contributions from the previous Chapter were included in this study. Even though we lost the *single-codeword training property*, a discussion on the training set is carried out, and the simulations, including various Polar and BCH codes, corroborate the decoder's performance and scalability properties.

Perspectives

Since the interest in machine learning applied to channel coding was renewed less than ten years ago [BCK18a; Gru+17; NBB16], the related research has consistently pushed the boundaries of high-performance and low-complexity decoders. In this thesis, we have attempted to do the same. Additionally, we identified the following research axes that were not included in this thesis, and could be explored in subsequent studies.

SVM training set reduction. Even if our proposed solution has the lowest complexity compared to those in previous works, the dataset size still carries an exponential growth, as the SVM classifier needs at least one sample for every possible codeword to produce the decision functions. SVMs have the property of only employing a small subset of the training samples —the so-called *support vectors*— to decide whether a sample belongs to a class or not. This property could be capitalized to reduce the training dataset to only the *necessary* codewords. Future studies may consider exploiting the structure of the error correction code in order to generate an SVM-based decoder that does not need a dataset composed of all the different codewords, but rather a small subset that may vary for every bit position.

Model-free and model-base convergence. The main difference between the two scalable neural-based approaches lies in the fact that the model-based network architecture is specific to a code (given by its Tanner graph), while the model-free has an independent structure. While this independence enables the use of more advanced deep learning-based methods, future works may explore how to exploit the code's structure in order to produce a *hybrid* approach that can reduce the number of weights in the network (e.g., by eliminating connections between independent nodes), while maintaining the learning potential of the model-free approach. We have already seen a glimpse of this phenomenon in the attention blocks of the ECCT and r-ECCT, which employ a mask that depends on the parity-check matrix. This does not result in a lower NN complexity but nonetheless uses the code's specific properties —in this case, the parity-check matrix— to *orient* the network's optimization process.

Effect of the training SNR. In this thesis, many network and training hyperparameters were selected *ad hoc*, e.g., the network size, activation functions, training SNR, learning rate, batch size, etc. Most of these parameters have an understandable impact on the optimization process and are shared with many other machine-learning applications: the network size impacts the learning potential, the learning rate changes the training time but can cause the optimization to diverge, the batch size influences the gradient computation and the training time, etc. In particular, the SNR proved to have a massive impact on the decoder's final performance, and it was very hard to predict the value that would yield the lowest error rates beforehand. Future works may study how to select the training SNR to simultaneously optimize the decoder's performance and generalization to all noise regimes, minimize training time, and avoid the grid-search that presently configures the SNR value selection process.

Autoencoders for end-to-end optimization. Thus far, we have worked exclusively with systems that decode already existing channel codes, such as Polar and BCH codes. However, further studies could explore employing autoencoders in order to simultaneously produce encoding and decoding functions that are optimal for the particular application. Autoencoders —such as Variational Autoencoders (VAE)— work by contracting the input space into a smaller representation that captures the main features of the input, followed by an expanding decoder to recover the original message. This framework should be inverted in order to resemble the channel encoding-decoding process.

Appendix A

Optimality of the SBND

In this Appendix, we prove the sufficient statistics theorem for the SBND framework, i.e., that using the absolute value $|\boldsymbol{y}|$ and the syndrome $H\boldsymbol{y}^b$ does not incur any loss of information with respect to the actual signal $\boldsymbol{y}$ when computing the posterior probability. Intuitively, this means that the value of $\boldsymbol{u}$ that maximizes the posterior probability $P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y})$, is also the value that maximizes $P_{\boldsymbol{E}^b_{\boldsymbol{u}}||\boldsymbol{Z}|,H\boldsymbol{Z}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b)$.

A.1 Proof of Theorem 3.2

Let us start by recalling the two claims of Lemma 1 in [BCK18b], regarding the framework of section 3.1.3. The proof is also added for self-containedness.

Lemma A.1. *Considering the framework from Section 3.1.3, the following claims hold:*

1. *There exists a matrix A with dimensions $k \times n$ such that $A\boldsymbol{c} = \boldsymbol{u}$ for all possible $\boldsymbol{u} \in \{0,1\}^k$ and its corresponding $\boldsymbol{c}$ through the code $\mathcal{C}$, that is, $f(\boldsymbol{c}) = A\boldsymbol{c}$ is a pseudo-inverse for $\mathcal{C}$.*
2. *Given a matrix $B = [H^T, A^T]^T$, then B has full column rank and is thus injective.*

Proof. Part 1 of the lemma follows from the properties of linear codes: its generator matrix G has full row rank and satisfies $\boldsymbol{c} = G^T\boldsymbol{u}$. Therefore, we can define A to be the left-inverse of G^T, and thus $A\boldsymbol{c} = (G^T)^{-1}_{\text{left}}\, G^T\boldsymbol{u} = \boldsymbol{u}$.

For part 2, let us start by assuming, without loss of generality, that H has full row rank (the linearly dependent rows are removed from the matrix). Then, the right-inverse of H exists and is denoted H^{-1}_{right}, of dimensions $n \times (n-k)$. Consider the following matrix product of two $n \times n$ matrices:

$$B \cdot [G^T, H^{-1}_{\text{right}}] = \begin{bmatrix} H \\ A \end{bmatrix} \cdot \begin{bmatrix} G^T & H^{-1}_{\text{right}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{n-k,k} & I_{n-k} \\ I_k & AH^{-1}_{\text{right}} \end{bmatrix}, \tag{A.1}$$

where $HG^T = 0$ follows from the properties of generator and parity-check matrices, and equalities $HH^{-1}_{\text{right}} = I_{n-k}$ and $AG^T = I_n$ follow from the definitions of said matrices. The resulting matrix has rank n (due to the identity matrices) and thus B must have rank n as well. □

Now, we need to provide a second result. Consider the following Lemma.

Lemma A.2. *For the random vectors $\boldsymbol{Y}$, $\boldsymbol{U}$ and $\tilde{\boldsymbol{U}}$ defined as in Section 3.2.1, the events $\mathcal{E}_1 = \{\boldsymbol{U} = \boldsymbol{U} | \boldsymbol{Y} = \boldsymbol{y}\}$ and $\mathcal{E}_2 = \{\boldsymbol{U} \oplus \tilde{\boldsymbol{U}} = \boldsymbol{u} \oplus \tilde{\boldsymbol{u}} | \boldsymbol{Y} = \boldsymbol{y}\}$ are equivalent.*

Proof. Considering that $\boldsymbol{Y} = \boldsymbol{y}$ and that $\tilde{\boldsymbol{u}}$ is a deterministic function of $\boldsymbol{y}$, it is trivial that $\mathcal{E}_1$ implies $\mathcal{E}_2$. Additionally, since $\tilde{\boldsymbol{u}} \oplus \tilde{\boldsymbol{u}} = \boldsymbol{0}$, then $\boldsymbol{U} \oplus \tilde{\boldsymbol{U}} \oplus \tilde{\boldsymbol{U}} = \boldsymbol{U}$. Therefore, the event $\mathcal{E}_2$ allows to unequivocally restore $\boldsymbol{U}$, and thus implying $\mathcal{E}_1$. □

With these two results, we can proceed to prove Theorem 3.2.

$$\begin{aligned}
P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) &\overset{(a)}{=} P_{\boldsymbol{U}\oplus\tilde{\boldsymbol{U}}|\boldsymbol{Y}}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, \boldsymbol{y}) \\
&\overset{(b)}{=} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,\boldsymbol{Y}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, \boldsymbol{y}^b) \\
&\overset{(c)}{=} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,B\boldsymbol{Y}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, B\boldsymbol{y}^b) \\
&\overset{(d)}{=} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,H\boldsymbol{Y}^b,A\boldsymbol{Y}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b, A\boldsymbol{y}^b) \\
&\overset{(e)}{=} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,H\boldsymbol{Z}^b,A\boldsymbol{C}\oplus A\boldsymbol{Z}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b, A\boldsymbol{y}^b) \\
&\overset{(f)}{=} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,H\boldsymbol{Z}^b,\boldsymbol{U}\oplus A\boldsymbol{Z}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b, A\boldsymbol{y}^b) \\
&\overset{(g)}{=} P_{\boldsymbol{E}^b_{\boldsymbol{u}}|\,|\boldsymbol{Z}|,H\boldsymbol{Z}^b}(\boldsymbol{u} \oplus \tilde{\boldsymbol{u}} \,|\, |\boldsymbol{y}|, H\boldsymbol{y}^b).
\end{aligned} \tag{A.2}$$

To obtain (a), we used Lemma 2. In (b), we used the definition of $\boldsymbol{E}^b_{\boldsymbol{u}}$ and decomposed the variable $\boldsymbol{Y}$ into its module and binary hard-decision, where $|\boldsymbol{Y}| = |\boldsymbol{Z}|$ by (3.5). In (c) and (d), the second claim of Lemma 1 was employed. In (e), we expressed $\boldsymbol{Y}^b$ as $\boldsymbol{C} \oplus \boldsymbol{Z}^b$, and exploited the validity of the codeword $\boldsymbol{C}$:

$$H\boldsymbol{Y}^b = H(\boldsymbol{C} \oplus \boldsymbol{Z}^b) = H\boldsymbol{Z}^b. \tag{A.3}$$

The pseudo-inverse $A\boldsymbol{C} = \boldsymbol{U}$ was employed to obtain (f). Finally, (g) made use of the following result:

$$\begin{aligned}
\boldsymbol{E}^b_{\boldsymbol{u}} &= \boldsymbol{U} \oplus A\boldsymbol{Y}^b \\
&= \boldsymbol{U} \oplus A(\boldsymbol{C} \oplus \boldsymbol{Z}^b) && \text{(A.4)} \\
&= \boldsymbol{U} \oplus \boldsymbol{U} \oplus A\boldsymbol{Z}^b && \text{(A.5)} \\
&= A\boldsymbol{Z}^b \;\perp\; \boldsymbol{U} \oplus A\boldsymbol{Z}^b, && \text{(A.6)}
\end{aligned}$$

where $\perp$ indicates independence between two random variables and $U_i \sim \mathrm{Ber}(1/2)$ $\forall i =$

$\{1, ..., k\}$. Given that $\boldsymbol{U} \oplus A\boldsymbol{Z}^b$ is independent of $\boldsymbol{E}^b_{\boldsymbol{u}}$, it can be removed from the conditional probability expression. This concludes the proof of Theorem 3.2.

A.2 Proof of Lemma 3.1

In this section, we will prove Lemma 3.1, which is a continuation of Theorem 3.2, where we specify the mathematical expression for the argmax of the posterior probability $P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y})$. Let us start by applying Bayes' formula on the posterior probability:

$$P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) = \frac{P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u})P_{\boldsymbol{U}}(\boldsymbol{u})}{P_{\boldsymbol{Y}}(\boldsymbol{y})}. \tag{A.7}$$

Now, recall that a uniform source is supposed, and thus $P_{\boldsymbol{U}}(\boldsymbol{u}) = 2^{-k}$, $\forall \boldsymbol{u} \in \{0,1\}^k$. Additionally, $\boldsymbol{u} = \tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}}$ as per (3.20). Finally, consider that $P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u}) = P_{\boldsymbol{Y}|\boldsymbol{X}(\boldsymbol{U})}(\boldsymbol{y}|\boldsymbol{x}(\boldsymbol{u}))$, where $\boldsymbol{x}(\boldsymbol{u})$ is the BPSK-modulated codeword corresponding to the message $\boldsymbol{u}$, i.e., $\boldsymbol{x} = 1 - 2\boldsymbol{u}$. All these things considered, let us now rewrite the posterior probability:

$$\begin{aligned}
P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) &\overset{(a)}{=} \frac{2^{-k}}{P_{\boldsymbol{Y}}(\boldsymbol{y})(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{y} - (1 - 2G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}}))||^2\right) \\
&\overset{(b)}{=} \frac{C}{P_{\boldsymbol{Y}}(\boldsymbol{y})} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^n \left(y_i^2 - 2y_i(1-2[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i) + (1-2[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i)^2\right)\right) \\
&\overset{(c)}{=} \frac{C}{P_{\boldsymbol{Y}}(\boldsymbol{y})} \exp\left(-\frac{1}{2\sigma^2}\left(n + \sum_{i=1}^n y_i^2 - 2y_i(1 - 2[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i)\right)\right) \\
&\overset{(d)}{=} \frac{C}{P_{\boldsymbol{Y}}(\boldsymbol{y})} \exp\left(-\frac{1}{2\sigma^2}\left(n + \sum_{i=1}^n y_i^2\right)\right) \exp\left(-\frac{1}{\sigma^2}\sum_{i=1}^n y_i(1 - 2[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i)\right),
\end{aligned}$$

where for each step, the following elements are used:

(a) The pdf of the Gaussian channel is inserted: $P_{\boldsymbol{Y}|\boldsymbol{U}}(\boldsymbol{y}|\boldsymbol{u}) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\frac{||\boldsymbol{y}-\boldsymbol{x}(\boldsymbol{u})||^2}{2\sigma^2}}$.

(b) A constant $C \triangleq \frac{2^{-k}}{(2\pi\sigma^2)^{n/2}}$ is defined, and the norm is developed. The notation $[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i$ denotes the ith element of the vector resulting from the operation $G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})$.

(c) The BPSK modulation is exploited, where $(1 - 2[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i)^2 = (\pm 1)^2 = 1$, and hence $\sum_{i=1}^n 1 = n$.

(d) The terms in the exponential are regrouped in order to separate the ones that depend on $\boldsymbol{u}$ from the independent ones.

Next, we apply the argmax operation, which allows us to remove the multiplicative elements that do not depend on the optimizing variable. Considering that $\frac{C}{P_{\boldsymbol{Y}}(\boldsymbol{y})} e^{-\frac{1}{2\sigma^2}\left(n + \sum_{i=1}^n y_i^2\right)} > 0$

is independent from $\boldsymbol{u}$, and that $\tilde{\boldsymbol{u}}$ is a deterministic function of $\boldsymbol{y}$, we have:

$$\begin{aligned}
\underset{\boldsymbol{u}}{\operatorname{argmax}}\, P_{\boldsymbol{U}|\boldsymbol{Y}}(\boldsymbol{u}|\boldsymbol{y}) &= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmax}} \sum_{i=1}^{n} y_i(1-2[G^T(\tilde{\boldsymbol{u}} \oplus \boldsymbol{e}^b_{\boldsymbol{u}})]_i) \\
&= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmax}} \sum_{i=1}^{n} y_i(1-2[G^T\tilde{\boldsymbol{u}} \oplus G^T\boldsymbol{e}^b_{\boldsymbol{u}})]_i) \\
&= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmax}} \sum_{i=1}^{n} y_i(1-2[G^T\tilde{\boldsymbol{u}}]_i)(1-2[G^T\boldsymbol{e}^b_{\boldsymbol{u}}]_i) \\
&= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmax}} \sum_{i=1}^{n} y_i(1-2[G^T\tilde{\boldsymbol{u}}]_i) - 2y_i(1-2[G^T\tilde{\boldsymbol{u}}]_i)[G^T\boldsymbol{e}^b_{\boldsymbol{u}}]_i \\
&= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmax}} \sum_{i=1}^{n} -2y_i(1-2[G^T\tilde{\boldsymbol{u}}]_i)[G^T\boldsymbol{e}^b_{\boldsymbol{u}}]_i \\
&= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmin}} \sum_{i=1}^{n} y_i(1-2[G^T\tilde{\boldsymbol{u}}]_i)[G^T\boldsymbol{e}^b_{\boldsymbol{u}}]_i \\
&= \underset{\boldsymbol{e}^b_{\boldsymbol{u}}}{\operatorname{argmin}} \sum_{i=1}^{n} y_i(1-2[G^T\operatorname{pinv}(\boldsymbol{y}^b)]_i)[G^T\boldsymbol{e}^b_{\boldsymbol{u}}]_i, \qquad \text{(A.8)}
\end{aligned}$$

where basic mathematical operations were employed, and the terms independent from $\boldsymbol{e}^b_{\boldsymbol{u}}$ were removed from the argmax, which was finally transformed into an argmin problem to resemble the OSD decoding procedure (which seeks to minimize the sum of the absolute value of the log-likelihoods of the bits that are flipped by the decoder). This yields the expression (A.8), which is independent of $\boldsymbol{u}$ as proved in Theorem 3.2. This concludes the proof.

Observation A.1. (A.8) depends on the generator matrix G instead of on the parity-check matrix H. This does not entail any problem, given that, as seen in Section 1.1.3, the generator matrix can be computed using the parity-check matrix.

Appendix B

Parity-check matrix analysis

In this Appendix, we provide the mathematical proof for the two results needed for Section 3.2.4, where the influence of the parity-check matrix is studied.

B.1 Proof of Theorem 3.3

Let H denote a valid parity-check matrix of size $n - k \times n$ for the code $\mathcal{C}$, i.e., $\boldsymbol{c} \in \mathcal{C}$ if and only if $H\boldsymbol{c} = \boldsymbol{0}$. The MI between the bitflip pattern and the syndrome can be expressed as follows:

$$I(\boldsymbol{E};\boldsymbol{S}) = \mathcal{H}(\boldsymbol{S}) - \underbrace{\mathcal{H}(\boldsymbol{S}|\boldsymbol{E})}_{=0} = \mathcal{H}(\boldsymbol{S}), \tag{B.1}$$

where $\mathcal{H}$ denotes the entropy and where we have used the fact that knowledge of the noise pattern provides full knowledge of its syndrome, and thus the conditional entropy $\mathcal{H}(\boldsymbol{S}|\boldsymbol{E})$ is null. The same is valid for $\tilde{\boldsymbol{S}}$. Next, let $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$ represent two different bitflip patterns that belong to the same coset of H, i.e., $H\boldsymbol{e_1} = H\boldsymbol{e_2} = \boldsymbol{s}$. We then have that:

$$H(\boldsymbol{e_1} \oplus \boldsymbol{e_2}) = \boldsymbol{s} \oplus \boldsymbol{s} = \boldsymbol{0}. \tag{B.2}$$

Consider now a different parity-check matrix $\tilde{H}$. Because they are both valid parity-check matrices for the code $\mathcal{C}$, they must share a kernel space, i.e., $\ker(H) = \ker(\tilde{H})$. Hence, due to the result in (B.2), we can deduce that:

$$\tilde{H}(\boldsymbol{e_1} \oplus \boldsymbol{e_2}) = \tilde{H}\boldsymbol{e_1} \oplus \tilde{H}\boldsymbol{e_2} = \tilde{\boldsymbol{s}}_{\boldsymbol{1}} \oplus \tilde{\boldsymbol{s}}_{\boldsymbol{2}} \equiv \boldsymbol{0}, \tag{B.3}$$

and thus $\tilde{\boldsymbol{s}}_{\boldsymbol{1}} = \tilde{\boldsymbol{s}}_{\boldsymbol{2}}$. This demonstrates that the coset distribution within a code remains unchanged regardless of the parity-check matrix employed. Thus, it follows from the definition of the entropy of a random variable that:

$$\mathcal{H}(\boldsymbol{S}) = \mathcal{H}(\tilde{\boldsymbol{S}}), \tag{B.4}$$

and consequently $I(\boldsymbol{E};\boldsymbol{S}) = I(\boldsymbol{E};\tilde{\boldsymbol{S}})$ as per (B.1). The MI is thereby proven to be invariant with respect to the choice of the parity-check matrix. This concludes the proof.

B.2 Proof of 3.2

For a bitflip vector $\boldsymbol{E} = (E_1, E_2, ..., E_n)$ and a syndrome $\boldsymbol{S} = (S_1, S_2, ..., S_{n-k})$, we have for $i \in \{1, ..., n\}$ and $j \in \{1, ..., n-k\}$:

$$I(E_i; S_j) = \mathcal{H}(S_j) - \mathcal{H}(S_j|E_i). \tag{B.5}$$

Observe that, if $H_{ij} = 0$, then S_j and E_i are independent and thus $\mathcal{H}(S_j|E_i) = \mathcal{H}(S_j)$, which yields an MI equal to 0. In the following, we consider only the case where $H_{ij} = 1$. Let N_j denote the number of ones –or *weight*– of the jth row of the parity-check matrix H. Additionally, if $\forall i \in \{1, ..., n\}$, E_i follows an independent Bernoulli distribution with bitflip probability p, then $P(S_j = 0)$ is equal to the probability of getting an *even* number of ones over N_j realizations of a Bernoulli random variable with the same flip probability p. That is:

$$P(S_j = 0) = \frac{1}{2} + \frac{1}{2}(1-2p)^{N_j} \triangleq \mathcal{E}(N_j), \tag{B.6}$$

where we have used that, after N_j Bernoulli realizations with probability p, $\mathrm{P}\{\text{even \# of ones}\} = \frac{1}{2} + \frac{1}{2}(1-2p)^{N_j}$. Therefore, the entropy of S_j boils down to:

$$\mathcal{H}(S_j) = \mathcal{H}_b(\mathcal{E}(N_j)), \tag{B.7}$$

where $\mathcal{H}_b(a) = -a\log_2 a - (1-a)\log_2(1-a)$ is the binary entropy function. Similarly:

$$\begin{aligned}\mathcal{H}(S_j|E_i) &= \sum_{e_i \in \{0,1\}} P(E_i = e_i)\mathcal{H}(S_j|E_i = e_i) \\ &= (1-p)\mathcal{H}_b(\mathcal{E}(N_j - 1)) + p\mathcal{H}_b(1 - \mathcal{E}(N_j - 1)) \\ &= \mathcal{H}_b(\mathcal{E}(N_j - 1)),\end{aligned} \tag{B.8}$$

where we have used that $\mathcal{H}_b(a) = \mathcal{H}_b(1-a)$. Finally, from (B.7) and (B.8), we have:

$$I(E_i; S_j) = [\mathcal{H}_b(\mathcal{E}(N_j)) - \mathcal{H}_b(\mathcal{E}(N_j - 1))]\, \mathbb{1}(H_{ij} = 1), \tag{B.9}$$

where we have added the condition that the entry H_{ij} is equal to 1. This concludes the proof.

Appendix C

Bit-Interleaved Coded Modulations

In this Appendix, we provide all the necessary proofs to apply the SBND approach in the BICM communication setting of Chapter 4.

C.1 Proof of LLR expressions

C.1.1 Proof of Lemma 4.2

Before proceeding with the proof of the lemma, let us first simplify further the approximate LLRs expression in the case of a PSK constellation. Let $y \in \mathbb{C}$, we have that:

$$l_s(y) \triangleq \gamma\big(\min_{x \in \mathcal{X}_1^s} ||y-x||^2 - \min_{x \in \mathcal{X}_0^s} ||y-x||^2\big) \tag{C.1}$$

$$= 2\gamma \mathrm{Re}(y(x_0^s(y) - x_1^s(y))^\star) \tag{C.2}$$

$$= 2\gamma \mathrm{Re}(y\Delta_s(y)), \tag{C.3}$$

where $x_0^s(y) = \mathrm{argmin}_{x \in \mathcal{X}_0^s} ||y-x||^2$ and $x_1^s(y) = \mathrm{argmin}_{x \in \mathcal{X}_1^s} ||y-x||^2$, and where for ease of notations, we introduce $\Delta_s(y) \triangleq (x_0^s(y) - x_1^s(y))^\star$.

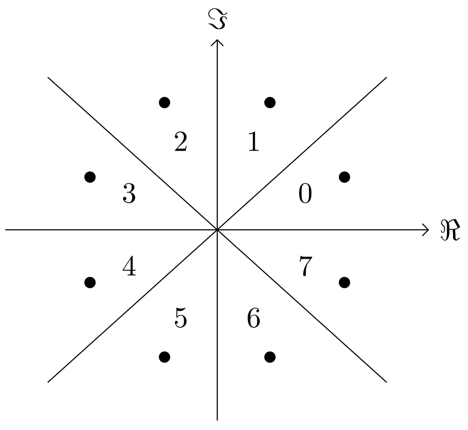
Next, to give analytical formulations of the LLRs of the different points y in the complex plane, we need to derive the closest symbols $x_0^s(y)$ and $x_1^s(y)$ for each of the decision regions of the constellation. Since the calculations are lengthy, yet easy to carry out, we will give the detailed proof for the case $s = 1$, and give a summary of the proof for the other values of s.

Let us give a closed-form expression of the three $l_s(y)$, $s \in [1:3]$. Figure C.1a gives a set of zones (labeled from 0 to 7) that differ in their associated hard decisions $x_0^s(y)$ and $x_1^s(y)$. Table C.1 gives the hard decisions associated with each of the different zones, along with their associated LLR expressions.

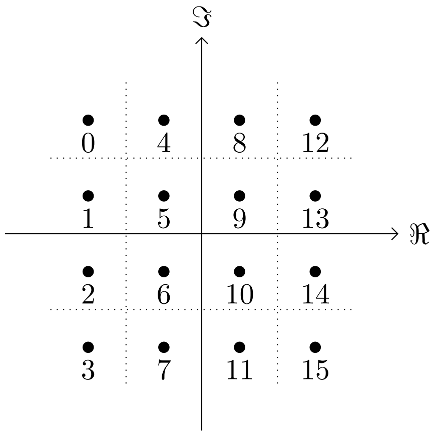
To end the proof for $l_1(y)$, note that zones 0, 3, 4, and 7 are defined by $|\mathrm{Im}(y)| \leq |\mathrm{Re}(y)|$, and that zones 1, 2, 5, and 6 are defined by $|\mathrm{Im}(y)| \geq |\mathrm{Re}(y)|$. Also, observe that for zones 1, 2, 5, and 6 the expression of $l_1(y)$ can be factorized as follows

$$l_1(y) = 2\sqrt{2}\gamma \left(\beta \mathrm{Im}(y) - \alpha \mathrm{sign}(\mathrm{Im}(y))|\mathrm{Re}(y)|\right) \tag{C.4}$$

$$= 2\sqrt{2}\gamma \mathrm{sign}(\mathrm{Im}(y)) \left(\beta|\mathrm{Im}(y)| - \alpha|\mathrm{Re}(y)|\right), \tag{C.5}$$



(a) 8-PSK.



(b) 16-QAM.

Figure C.1: Zone partitioning for the two considered constellations

Zone	$x_0^1(y)$	$x_1^1(y)$	$\Delta_1(y)$	$l_1(y)$
0	x_0	x_7	$-2\alpha j$	$4\gamma\alpha\text{Im}(y)$
1	x_1	x_7	$\sqrt{2}e^{-j\frac{5\pi}{8}}$	$2\sqrt{2}\gamma(-\alpha\text{Re}(y)+\beta\text{Im}(y))$
2	x_2	x_4	$\sqrt{2}e^{-j\frac{3\pi}{8}}$	$2\sqrt{2}\gamma(\alpha\text{Re}(y)+\beta\text{Im}(y))$
3	x_3	x_4	$-2\alpha j$	$4\gamma\alpha\text{Im}(y)$
4	x_3	x_4	$-2\alpha j$	$4\gamma\alpha\text{Im}(y)$
5	x_3	x_5	$\sqrt{2}e^{-j\frac{5\pi}{8}}$	$2\sqrt{2}\gamma(-\alpha\text{Re}(y)+\beta\text{Im}(y))$
6	x_0	x_6	$\sqrt{2}e^{-j\frac{3\pi}{8}}$	$2\sqrt{2}\gamma(\alpha\text{Re}(y)+\beta\text{Im}(y))$
7	x_0	x_7	$-2\alpha j$	$4\gamma\alpha\text{Im}(y)$

Table C.1: Hard decisions and $l_1(y)$ for the 8-PSK.

which completes the proof for $l_1(y)$. The expression for $l_2(y)$ can be obtained easily by swapping $\mathrm{Re}(y)$ and $\mathrm{Im}(y)$ in the expression of $l_1(y)$ given that the decision region $\mathcal{X}_0^2$ is a simple $\pi/2$-rotation of the decision region $\mathcal{X}_0^1$, and given the invariance of the constellation to a $\pi/2$-rotation. As for the expression of $l_3(y)$, it can be obtained easily following similar lines as $l_1(y)$ and is omitted here for conciseness. □

C.1.2 Proof of Lemma 4.3

In the case of a 16-QAM, one can show that the approximate LLRs are given by:

$$\begin{aligned} l_s(y) &= 2\gamma \mathrm{Re}(y(x_0^s(y) - x_1^s(y))^\star) + \gamma(|x_1^s(y)|^2 - |x_0^s(y)|^2) && \text{(C.6)} \\ &= 2\gamma \mathrm{Re}(y\Delta_s(y)) + \gamma d_s(y), && \text{(C.7)} \end{aligned}$$

where, for ease of notation, we further define $d_s(y) \triangleq |x_1^s(y)|^2 - |x_0^s(y)|^2$, while x_1^s, x_0^s and $\Delta_s(y)$ are as defined in the proof of Lemma 4.2 for the 8-PSK case. Similarly to the 8-PSK case, we can identify 16 decision zones that differ in their associated hard decisions as depicted in Figure C.1b.

Similarly to the proof of Lemma 4.2, we will make a detailed proof for $l_1(y)$, and give only the outline for the remaining expressions of $l_2(y)$, $l_3(y)$, and $l_4(y)$. Let us consider $s = 1$, and let us give the expression of $l_1(y)$. Table C.2 gives a detailed expression of $\Delta_1(y)$, $d_1(y)$, and $l_1(y)$ for each of the 16 zones, that were further grouped to make notations more compact.

Zones	$\Delta_1(y)$	$d_1(y)$	$l_1(y)$
$0 \to 3$	$-2d$	$-2d^2$	$-2\gamma d(2\mathrm{Re}(y) + d)$
$4 \to 11$	$-d$	0	$-2\gamma d\mathrm{Re}(y)$
$12 \to 15$	$-2d$	$2d^2$	$-2\gamma d(2\mathrm{Re}(y) - d)$

Table C.2: Hard decisions and $l_1(y)$ for the 16-QAM.

Next, noting that zones $0 \to 3$ and $12 \to 15$ are defined by $|\mathrm{Re}(y)| > d$, while zones $4 \to 11$ are defined by $|Re(y)| \leq d$, and noting that for zones $0 \to 3$ and $12 \to 15$, we have that

$$\begin{aligned} l_1(y) &= -2\gamma d(2\mathrm{Re}(y) - \mathrm{sign}(\mathrm{Re}(y))d) && \text{(C.8)} \\ &= -2\gamma d\,\mathrm{sign}(\mathrm{Re}(y))(2|\mathrm{Re}(y)| - d), && \text{(C.9)} \end{aligned}$$

which completes the proof for $l_1(y)$. From the expression of $l_1(y)$, the expression of $l_3(y)$ follows by replacing $\mathrm{Re}(y)$ with $-\mathrm{Im}(y)$. As for $l_2(y)$ and $l_4(y)$, their expressions are easy to obtain following similar lines as $l_1(y)$ and are omitted here for compactness. □

C.2 Proof of Theorem 4.1

First, note that since each L^b is a deterministic function of L, and given that $P_{\boldsymbol{L}|\boldsymbol{C}}$ is a memoryless channel, then so is $P_{\boldsymbol{L}^b|\boldsymbol{C}}$, i.e., for all $\boldsymbol{l}_b, \boldsymbol{c} \in \{0,1\}^n$:

$$P_{\boldsymbol{L}^b|\boldsymbol{C}}(\boldsymbol{l}^b|\boldsymbol{c}) = \prod_{i=1}^{n} P_{L^b|C}(l_i^b|c_i). \tag{C.10}$$

To proceed with the proof, we will first prove that $P_{L^b|C}(0|0) = P_{L^b|C}(1|1)$. Consider the following result:

$$\begin{aligned}
P_{L^b|C}(0|0) &= \frac{1}{m}\sum_{s=1}^{m} P_{L^b|C,S}(0|0,s) && \text{(C.11)}\\
&\overset{(a)}{=} \frac{1}{m|\mathcal{X}_0^s|}\sum_{s=1}^{m}\sum_{x\in\mathcal{X}_0^s} P_{L^b|X,S}(0|x,s) && \text{(C.12)}\\
&= \frac{1}{m|\mathcal{X}_0^s|}\sum_{s=1}^{m}\sum_{x\in\mathcal{X}_0^s} \int_{\mathbb{C}} P_{L^b,Y|X,S}(0,y|x,s)dy && \text{(C.13)}\\
&\overset{(b)}{=} \frac{1}{m|\mathcal{X}_0^s|}\sum_{s=1}^{m}\sum_{x\in\mathcal{X}_0^s} \int_{\mathcal{Y}_0^s} P_{Y|X,S}(y|x,s)dy && \text{(C.14)}\\
&\overset{(c)}{=} \frac{1}{m|\mathcal{X}_0^s|}\sum_{s=1}^{m}\sum_{x\in\mathcal{X}_0^s} \int_{\mathcal{Y}_0^s} P_{Y|X}(y|x)dy, && \text{(C.15)}
\end{aligned}$$

where we define the decision region $\mathcal{Y}_0^s \triangleq \{y \in \mathbb{C}, l_s(y) > 0\}$, where $l_s(y)$ is the LLR of the s-th bit in the transmitted symbol, and where (a) follows the uniformity of the constellation, while (b) follows from the Markov chain $C \leftrightarrow (X,S) \leftrightarrow L^b$, and (c) from the Markov chain $S \leftrightarrow X \leftrightarrow Y$.

Let us then consider the 8-PSK constellation with Gray mapping of Figure C.2. Taking into account the expressions of the LLRs $l_s(y)$ in Lemma 4.2, we give in Figure C.2 the decision regions $\mathcal{Y}_0^s$ for $s \in [1:3]$.

Let us now consider the case $s = 1$. We have that $\mathcal{X}_0^1 = -(\mathcal{X}_1^1)^\star$ and $\mathcal{Y}_0^1 = -(\mathcal{Y}_1^1)^\star$. Thus, by exploiting one of the symmetries of the normal Gaussian probability distribution $P_{Y|X}(y|x) = P_{Y|X}(-y^\star|-x^\star)$, one can easily prove that

$$\sum_{x\in\mathcal{X}_0^1}\int_{\mathcal{Y}_0^1} P_{Y|X}(y|x)dy = \sum_{x\in\mathcal{X}_1^1}\int_{\mathcal{Y}_1^1} P_{Y|X}(y|x)dy. \tag{C.16}$$

Similar results can be proved for $s = 2$ by noticing that $\mathcal{X}_0^2 = (\mathcal{X}_1^2)^\star, \mathcal{Y}_0^2 = (\mathcal{Y}_1^2)^\star$ and using the property $P_{Y|X}(y|x) = P_{Y|X}(y^\star|x^\star)$. Finally, for $s = 3$, note that $\mathcal{X}_0^3 = \mathcal{X}_1^3 e^{j\pi/2}$ and $\mathcal{Y}_0^3 = \mathcal{Y}_1^3 e^{j\pi/2}$ along with the symmetry $P_{Y|X}(ye^{j\phi}|xe^{j\phi}) = P_{Y|X}(y^\star|x^\star)$ for all ϕ yields the same result as (C.16). Hence, recalling that $|\mathcal{X}_0^s| = |\mathcal{X}_1^s|$, we have:

$$P_{L^b|C}(0|0) = \frac{1}{m|\mathcal{X}_0^s|}\sum_{s=1}^{m}\sum_{x\in\mathcal{X}_0^s}\int_{\mathcal{Y}_0^s} P_{Y|X}(y|x)dy \tag{C.17}$$

$$= \frac{1}{m|\mathcal{X}_1^s|}\sum_{s=1}^{m}\sum_{x\in\mathcal{X}_1^s}\int_{\mathcal{Y}_1^s} P_{Y|X}(y|x)dy \tag{C.18}$$

$$= P_{L^b|C}(1|1). \tag{C.19}$$

Hence, for the 8-PSK, the channel $P_{L^b|C}$ is a binary symmetric channel, with crossover probability q given by:

$$q = P_{L^b|C}(1|0) = \frac{1}{m}\sum_{s=1}^{m} P^s_{L^b|C}(1|0). \tag{C.20}$$

Concerning the 16-QAM, we represent in Figure C.3 the decision regions $\mathcal{Y}_1^s$ for $s \in [1:4]$. It can be easily seen that, for $s = 1$ and $s = 3$, using the symmetries of $\mathcal{X}_c^s$ and of the Gaussian distribution, one can write that:

$$\sum_{x\in\mathcal{X}_0^s}\int_{\mathcal{Y}_0^s} P_{Y|X}(y|x)dy = \sum_{x\in\mathcal{X}_1^s}\int_{\mathcal{Y}_1^s} P_{Y|X}(y|x)dy. \tag{C.21}$$

However, for $s = 2$ and $s = 4$, the previous equality does not hold in the sense that the integration over $\mathcal{Y}_1^s$ entails a larger clipping of the Gaussian distribution than the integration over $\mathcal{Y}_0^s$. However, if the SNR is not too low –e.g., SNR ≥ 0 dB for a normalized 16-QAM constellation– one can show that both integrations yield the same probability. The proof follows then similarly to the 8-PSK case. □

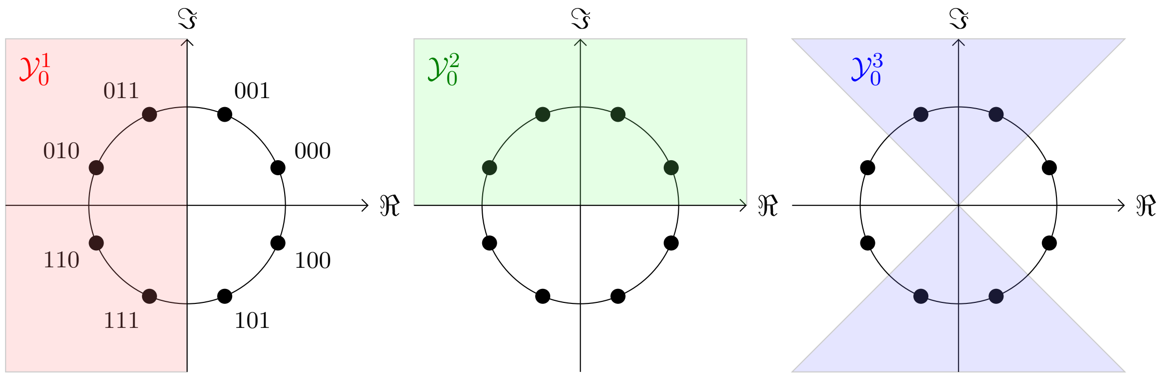


Figure C.2: Decision regions of the 8-PSK constellation.

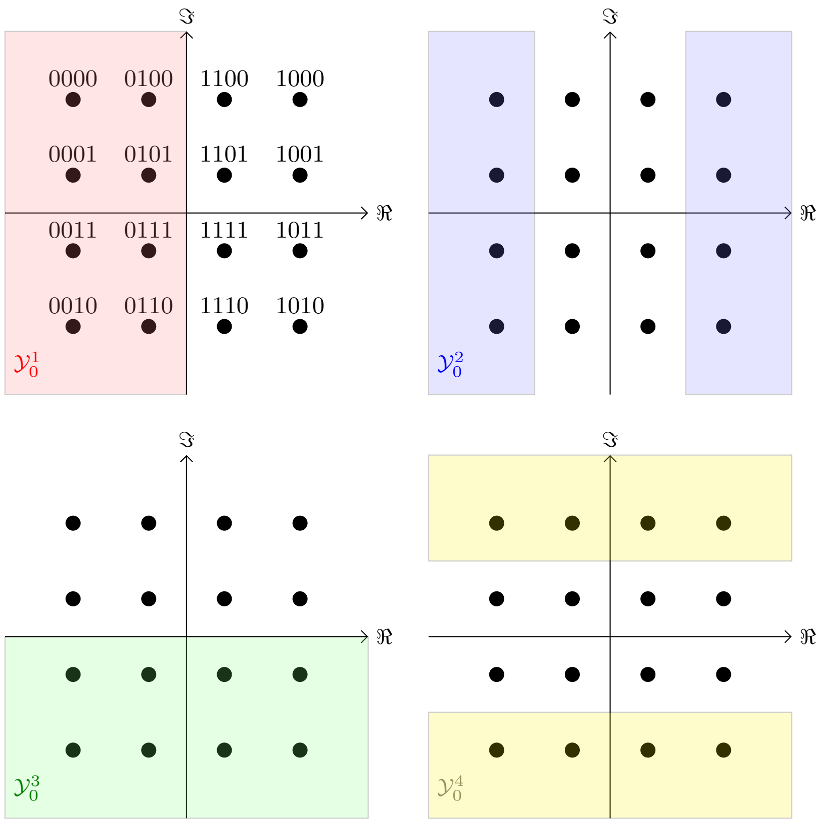


Figure C.3: Decision regions of the 16-QAM constellation.

C.3 Proof of Theorem 4.2

Let us start with two properties of an (n, k) linear block code.

(i) The pseudo-inverse of a code $p_{inv}(\cdot)$ is defined by a $k \times n$ matrix A such that $A\boldsymbol{c} = \boldsymbol{u}$;

(ii) the matrix $B = [H^T, A^T]$, where H is the parity matrix of the code, is full rank, thus, invertible.

Next, we can write for $\boldsymbol{e}_{\boldsymbol{u}}^b \in \{0,1\}^k$ and $\boldsymbol{l} \in \mathbb{R}^n$:

$$P_{\boldsymbol{E}_{\boldsymbol{u}}^b|\boldsymbol{L}}(\boldsymbol{e}_{\boldsymbol{u}}^b|\boldsymbol{l}) = P_{\boldsymbol{E}_{\boldsymbol{u}}^b||\boldsymbol{L}|,\boldsymbol{L}^b}(\boldsymbol{e}_{\boldsymbol{u}}^b|\,|\boldsymbol{l}|, \boldsymbol{l}^b) \tag{C.22}$$

$$= P_{\boldsymbol{E}_{\boldsymbol{u}}^b||\boldsymbol{L}|,H\boldsymbol{L}^b,A\boldsymbol{L}^b}(\boldsymbol{e}_{\boldsymbol{u}}^b|\,|\boldsymbol{l}|, H\boldsymbol{l}^b, A\boldsymbol{l}^b), \tag{C.23}$$

where we have used the fact that $B = [H^T, A^T]$ is invertible. Next, recalling the result of Theorem 4.1 in (4.20), we have that:

$$A\boldsymbol{L}^b = A\boldsymbol{C} \oplus \boldsymbol{E}_{\boldsymbol{u}}^b = \boldsymbol{U} \oplus \boldsymbol{E}_{\boldsymbol{u}}^b. \tag{C.24}$$

Since, $\boldsymbol{U}$ is i.i.d following a Bern(0.5) distribution, and since $\boldsymbol{E}_{\boldsymbol{u}}^{b}$ is independent of $\boldsymbol{U}$, then $A\boldsymbol{L}^{b}$ is Bern(0.5) and is independent of $\boldsymbol{E}_{\boldsymbol{u}}^{b}$. Hence, it follows that:

$$P_{\boldsymbol{E}_{\boldsymbol{u}}^{b}||\boldsymbol{L}|,H\boldsymbol{L}^{b},A\boldsymbol{L}^{b}}(\boldsymbol{e}_{\boldsymbol{u}}^{b}||\boldsymbol{l}|,H\boldsymbol{l}^{b},A\boldsymbol{l}^{b}) = P_{\boldsymbol{E}_{\boldsymbol{u}}^{b}||\boldsymbol{L}|,H\boldsymbol{L}^{b}}(\boldsymbol{e}_{\boldsymbol{u}}^{b}||\boldsymbol{l}|,H\boldsymbol{l}^{b}). \tag{C.25}$$

Finally, by marginalizing both the right-hand side and left-hand side with respect to the $k-1$ variables $(E_{u,1}^{b},\ldots,E_{u,i-1}^{b},E_{u,i+1}^{b},\ldots,E_{u,k}^{b})$ for all $i \in [1:k]$, concludes the proof. □

C.4 Proof of Lemma 4.4

C.4.1 8-PSK reliabilities invariances $|l_s|$

Let us consider the LLRs of the 8-PSK constellation of Lemma 4.2 and analyze the LLR of the 1st and 2nd bit-positions of the labeling $s=1$ and $s=2$. Let $y \in \mathbb{C}$, we have that:

$$|l_1(y)| = \begin{cases} 2\gamma\alpha|\mathrm{Im}(y)| & \text{if } |\mathrm{Im}(y)| \le |\mathrm{Re}(y)| \\ 2\sqrt{2}\gamma\ \ \big|\beta|\mathrm{Im}(y)| - \alpha|\mathrm{Re}(y)|\big| & \text{else} \end{cases}$$

$$|l_2(y)| = \begin{cases} 2\gamma\alpha|\mathrm{Re}(y)| & \text{if } |\mathrm{Re}(y)| \le |\mathrm{Im}(y)| \\ 2\sqrt{2}\gamma\ \ \big|\beta|\mathrm{Re}(y)| - \alpha|\mathrm{Im}(y)|\big| & \text{else} \end{cases}$$

Note that, irrespective of whether $|\mathrm{Im}(y)| \le |\mathrm{Re}(y)|$ or $|\mathrm{Im}(y)| > |\mathrm{Re}(y)|$, it follows that

$$|l_1(y)| = |l_1(-y)| = |l_1(y^\star)| = |l_1(-y^\star)| \tag{C.26}$$
$$|l_2(y)| = |l_2(-y)| = |l_2(y^\star)| = |l_2(-y^\star)|. \tag{C.27}$$

Assume that $y = x + w$ and let us consider the labels of the symbols of the constellation as shown in Figure 4.5. Then, if $x = x_0$ or $x = x_1$, the result follows. If $x = x_2$, then

$$-y^\star = -x_2^\star - w^\star = x_1 - w^\star, \tag{C.28}$$

which, by the circularity of the random noise W, i.e., $P_W(w) = P_W(-w^\star)$ and by (C.26) proves our claim for both $s=1$ and $s=2$. If $x = x_3$, then, since $x_0 = -x_3^\star$, $-y^\star = x_0 - w^\star$, which proves similarly the claim. The same reasoning can be iterated for all possibly transmitted symbols x, showing that either y, $-y$, $y^\star$ or $-y^\star$ would correspond to a transmitted symbol equal to x_0 or x_1 with the same reliability value $|l_s|$ for $s=1$ and $s=3$, and with the same noise probability since $P_W(w) = P_W(-w) = P_W(w^\star) = P_W(-w^\star)$.

As for $s=3$, recalling the result of Lemma 4.2, one can write that, for all $y \in \mathbb{C}$:

$$|l_3(y)| = 2\gamma(\beta-\alpha)\big||\mathrm{Im}(y)| - |\mathrm{Re}(y)|\big|. \tag{C.29}$$

In this case, we can show that:

$$|l_3(y)| = |l_3(-y)| = |l_3(y^\star)| = |l_3(-y^\star)|$$

$$= |l_3(jy)| = |l_3(-jy)| = |l_3(jy^\star)| = |l_3(-jy^\star)|, \tag{C.30}$$

where $j^2 = -1$. This implies a stronger result than for $s = 1$ and $s = 2$ in that all received noisy symbols y could be mapped to a noisy symbol obtained from the transmission of x_0 while maintaining the same observed reliability $|l_3(y)|$.

To conclude, for the 8-PSK, all possible reliabilities $|l_s|$ can be generated from the transmission of x_0 and x_1. $\square$

C.4.2 16-QAM reliabilities invariances $|l_s|$

Let us consider the 16-QAM constellation of Figure 4.5 and let $y \in \mathbb{C}$. Following similar lines as for the 8-PSK, let us recall the expressions of the LLRs of Lemma 4.3. We have that, for $s = 1$ and $s = 3$,

$$|l_1(y)| = \begin{cases} 2\gamma d|\mathrm{Re}(y)| & \text{if } |\mathrm{Re}(y)| \leq d \\ 2\gamma d \, |2|\mathrm{Re}(y)| - d| & \text{else,} \end{cases} \tag{C.31}$$

$$|l_3(y)| = \begin{cases} 2\gamma d|\mathrm{Im}(y)| & \text{if } |\mathrm{Im}(y)| \leq d \\ 2\gamma d \, |2|\mathrm{Im}(y)| - d| & \text{else.} \end{cases} \tag{C.32}$$

It can be shown from the above expressions that whatever the interval to which $|\mathrm{Re}(y)|$ and $|\mathrm{Im}(y)|$ belong, we have

$$\begin{aligned} |l_1(y)| &= |l_1(y + jz)| = |l_1(-y + jz)| = |l_1(y^\star + jz)| \\ |l_3(y)| &= |l_3(y + z)| = |l_3(-y + z)| = |l_3(y^\star + z)|, \end{aligned}$$

for any $z \in \mathbb{R}$. Hence, assuming that $y = x + w$, if $x = x_0$ or $x = x_1$ or $x = x_4$, the result is trivial for $s = 1$ and $s = 3$. Otherwise, if for instance $x = x_2$, then observing that

$$y + 2jd = x_2 + 2jd + w = x_0 + w, \tag{C.33}$$

the result follows. Similarly, if $x = x_5$, then noting that $x_5 = x_4 - jd$ allows to write that $y - jd = x_4 + w$. Hence, one can follow similar lines of proof for all possibly transmitted symbols of the 16 QAM constellation by resorting either to $y + kjd$ or $-y^\star + kjd$ for $s = 1$, or $y + kd$ or $y^\star + kd$ for $s = 3$, where $k \in [-3 : 3]$. As for $s = 2$, and $s = 4$, having that

$$|l_2(y)| = 2\gamma d \, ||\mathrm{Re}(y)| - d| \tag{C.34}$$

$$|l_4(y)| = 2\gamma d \, ||\mathrm{Im}(y)| - d| \, , \tag{C.35}$$

one could show the result of the theorem by using the three symbols x_0, x_1, and x_4 following similar lines as the case of $s = 1$ and $s = 3$. $\square$

Bibliography

[3GP18] 3GPP. “ETSI TS 123 501.” In: *5G; System Architecture for the 5G System* (2018) (cit. on p. 1).

[3GP20] 3GPP. “ETSI TS 138 212.” In: *5G NR Multiplexing and channel coding* (2020) (cit. on p. 24).

[Ahm+20] Ijaz Ahmad et al. “Machine Learning Meets Communication Networks: Current Trends and Future Challenges.” In: *IEEE Access* 8 (2020), pp. 223418–223460 (cit. on p. 1).

[Alv08] Alex Alvarado. *On Bit-interleaved Coded Modulation with QAM Constellations.* English. Other. 2008 (cit. on pp. 7, 73, 74, 76).

[AML12] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.T. Lin. *Learning from data.* Vol. 4. AMLBook New York, 2012 (cit. on pp. 3, 28).

[APP20] Sami Akın, Maxim Penner, and Jürgen Peissig. *Joint Channel Estimation and Data Decoding using SVM-based Receivers.* 2020 (cit. on pp. 3, 33).

[Ari09] Erdal Arikan. “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels.” In: *IEEE Transactions on Information Theory* 55.7 (July 2009), pp. 3051–3073 (cit. on pp. 2, 5, 6, 15, 16, 24).

[BB89] J. Bruck and M. Blaum. “Neural Networks, Error-Correcting Codes, and Polynomials Over the Binary n-cube.” In: *IEEE Transactions on Information Theory* 35.5 (1989), pp. 976–987 (cit. on p. 2).

[BBB15] Alexios Balatsoukas-Stimming, Mani Bastani Parizi, and Andreas Burg. “LLR-Based Successive Cancellation List Decoding of Polar Codes.” In: *IEEE Transactions on Signal Processing* 63.19 (Oct. 2015), pp. 5165–5179 (cit. on p. 23).

[BCK18a] Amir Bennatan, Yoni Choukroun, and Pavel Kisilev. “Deep Learning for Decoding of Linear Codes - A Syndrome-Based Approach.” In: *2018 IEEE International Symposium on Information Theory (ISIT).* IEEE, June 2018 (cit. on pp. 5, 6, 45–47, 49–51, 54, 55, 57, 58, 64, 65, 67, 71–73, 77–79, 87, 92).

[BCK18b] Amir Bennatan, Yoni Choukroun, and Pavel Kisilev. *Deep Learning for Decoding of Linear Codes - A Syndrome-Based Approach.* 2018 (cit. on pp. 50, 95).

[BCL21] Valerio Bioglio, Carlo Condo, and Ingmar Land. “Design of Polar Codes in 5G New Radio.” In: *IEEE Communications Surveys & Tutorials* 23.1 (2021), pp. 29–40 (cit. on p. 24).

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. “A Training Algorithm for Optimal Margin Classifiers.” In: *Proceedings of the fifth annual workshop on Computational learning theory.* COLT92. ACM, July 1992 (cit. on pp. 3, 28).

[Bla19] Mario Blaum. *A Short Course on Error-Correcting Codes.* 2019 (cit. on p. 11).

[BR60] R.C. Bose and D.K. Ray-Chaudhuri. "On a class of error correcting binary group codes." In: *Information and Control* 3.1 (Mar. 1960), pp. 68–79 (cit. on p. 15).

[Cha72] David Chase. "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information." In: *IEEE Transactions on Information Theory* 18.1 (Jan. 1972), pp. 170–182 (cit. on p. 56).

[Che+23] Qi Chen et al. "A Novel Labeling Scheme for Neural Belief Propagation in Polar Codes." In: *2023 International Wireless Communications and Mobile Computing (IWCMC).* IEEE, June 2023 (cit. on p. 5).

[Cho+14] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Association for Computational Linguistics, 2014 (cit. on p. 51).

[Cho+15] Fran**c**cois Chollet et al. *Keras.* 2015 (cit. on pp. 64, 85).

[CTB98] G. Caire, G. Taricco, and E. Biglieri. "Bit-Interleaved Coded Modulation." In: *IEEE Transactions on Information Theory* 44.3 (May 1998), pp. 927–946 (cit. on pp. 7, 73, 74, 76).

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks." In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297 (cit. on pp. 3, 28).

[CW22a] Yoni Choukroun and Lior Wolf. *Error Correction Code Transformer.* 2022 (cit. on pp. 6, 46, 52, 53, 55, 64, 65, 67, 69, 71, 72, 79, 87).

[CW22b] Yoni Choukroun and Lior Wolf. "Error Correction Code Transformer." In: *Advances in Neural Information Processing Systems.* Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 38695–38705 (cit. on pp. 6, 57).

[CW23] Yoni Choukroun and Lior Wolf. "Denoising Diffusion Error Correction Codes." In: *The Eleventh International Conference on Learning Representations.* 2023 (cit. on pp. 6, 46, 55, 79).

[DAM22] Ken R. Duffy, Wei An, and Muriel Medard. "Ordered Reliability Bits Guessing Random Additive Noise Decoding." In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 4528–4542 (cit. on p. 89).

[DB23] Gastón De Boni Rovella and Meryem Benammar. "Improved Syndrome-based Neural Decoder for Linear Block Codes." In: *GLOBECOM 2023 - 2023 IEEE Global Communications Conference.* IEEE, Dec. 2023 (cit. on pp. 47, 54).

[DeB+23] Gastón De Boni Rovella et al. "SVM pour la démodulation et le décodage conjoints." In: *GRETSI* (Aug. 2023) (cit. on p. 28).

[DeB+24a] Gastón De Boni Rovella et al. "On the Optimality of Support Vector Machines for Channel Decoding." In: *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit).* IEEE, June 2024 (cit. on pp. 28, 35).

[DeB+24b] Gastón De Boni Rovella et al. “Optimizing the Parity-Check Matrix for Syndrome-Based Neural Decoders (**submitted**).” In: *IEEE Communications Letters* (2024) (cit. on p. 54).

[DeB+24c] Gastón De Boni Rovella et al. “Scalable Syndrome-based Neural Decoders for Bit-Interleaved Coded Modulations.” In: *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. IEEE, May 2024, pp. 341–346 (cit. on p. 74).

[DeB+24d] Gastón De Boni Rovella et al. “Syndrome-Based Neural Decoding for Higher-Order Modulations (**submitted**).” In: *IEEE Transactions on Communications* (2024) (cit. on pp. 47, 74).

[DeB+25] Gastón De Boni Rovella et al. “Bitwise Approach for Optimal SVM-based Decoding (**submitted**).” In: *EURASIP Journal on Wireless Communications and Networking* (2025) (cit. on p. 28).

[DH10] Robert Daniels and Robert W. Heath. “Online Adaptive Modulation and Coding with Support Vector Machines.” In: *2010 European Wireless Conference (EW)*. IEEE, 2010 (cit. on pp. 3, 28).

[DS17] Rahul Dey and Fathi M. Salem. “Gate-variants of Gated Recurrent Unit (GRU) neural networks.” In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, Aug. 2017 (cit. on p. 58).

[Eld+22] Yonina C Eldar et al. *Machine learning and wireless communications*. Cambridge University Press, 2022 (cit. on p. 1).

[FL95] M.P.C. Fossorier and Shu Lin. “Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics.” In: *IEEE Transactions on Information Theory* 41.5 (1995), pp. 1379–1396 (cit. on pp. 21, 23, 85, 89).

[Gal63] Robert G. Gallager. *Low-Density Parity-Check Codes*. The MIT Press, 1963 (cit. on p. 5).

[Gar+06] M.J.F.-G. Garcia et al. “Support Vector Machines for Robust Channel Estimation in OFDM.” In: *IEEE Signal Processing Letters* 13.7 (July 2006), pp. 397–400 (cit. on p. 3).

[GB08] Michael Grant and Stephen Boyd. “Graph implementations for nonsmooth convex programs.” In: *Recent Advances in Learning and Control*. Ed. by V. Blondel, S. Boyd, and H. Kimura. Lecture Notes in Control and Information Sciences. `http://stanford.edu/~boyd/graph_dcp.html`. Springer-Verlag Limited, 2008, pp. 95–110 (cit. on p. 39).

[GB14] Michael Grant and Stephen Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. `https://cvxr.com/cvx`. Mar. 2014 (cit. on p. 39).

[Gia+18] E. Giacoumidis et al. “Unsupervised Support Vector Machines for Nonlinear Blind Equalization in CO-OFDM.” In: *IEEE Photonics Technology Letters* 30.12 (June 2018), pp. 1091–1094 (cit. on p. 3).

[GK99] Xiaohong Gong and A. Kuh. “Support vector machine for multiuser detection in CDMA communications.” In: *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers (Cat. No.CH37020)*. ACSSC-99. IEEE, 1999 (cit. on p. 3).

[Gru+17] Tobias Gruber et al. “On deep learning-based channel decoding.” In: *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, Mar. 2017 (cit. on pp. 2, 4, 92).

[HC06] Thomas Halford and Keith Chugg. “Random Redundant Soft-In Soft-Out Decoding of Linear Block Codes.” In: *2006 IEEE International Symposium on Information Theory*. IEEE, July 2006 (cit. on p. 71).

[HDL19] Tiep M. Hoang, Trung Q. Duong, and Sangarapillai Lambotharan. “Secure Wireless Communication Using Support Vector Machines.” In: *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, June 2019 (cit. on p. 3).

[Hel+19] Michael Helmling et al. *Database of Channel Codes and ML Simulation Results*. `www.uni-kl.de/channel-codes`. 2019 (cit. on pp. 39, 62, 65, 71).

[Hil90] Raymond Hill. *A First Course in Coding Theory*. Oxford University Press, USA, 1990, p. 264 (cit. on pp. 14, 15).

[HL02] Chih-Wei Hsu and Chih-Jen Lin. “A Comparison of Methods for Multiclass Support Vector Machines.” In: *IEEE Transactions on Neural Networks* 13.2 (Mar. 2002), pp. 415–425 (cit. on pp. 33, 34).

[Hoc59] Alexis Hocquenghem. “Codes correcteurs d’erreurs.” In: *Chiffers* 2 (1959), pp. 147–156 (cit. on p. 15).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory.” In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on p. 51).

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators.” In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366 (cit. on pp. 4, 83).

[Jia+17] Chunxiao Jiang et al. “Machine Learning Paradigms for Next-Generation Wireless Networks.” In: *IEEE Wireless Communications* 24.2 (Apr. 2017), pp. 98–105 (cit. on p. 1).

[KB08] J.W.H. Kao and S. M. Berber. “Error Control Coding Based on Support Vector Machine.” In: (2008) (cit. on pp. 3, 28, 33, 34, 36, 40).

[Kha+19] Faisal Nadeem Khan et al. “An Optical Communication’s Perspective on Machine Learning and Its Applications.” In: *Journal of Lightwave Technology* 37.2 (Jan. 2019), pp. 493–516 (cit. on p. 1).

[KP20] E. Kavvousanos and V. Paliouras. “An Iterative Approach to Syndrome-based Deep Learning Decoding.” In: *2020 IEEE Globecom Workshops (GC Wkshps*. IEEE, Dec. 2020 (cit. on pp. 6, 46, 79).

[LG18] Loren Lugosch and Warren J. Gross. “Learning from the Syndrome.” In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, Oct. 2018 (cit. on p. 5).

[LHL05] Cheng-Jian Lin, Shang-Jin Hong, and Chi-Yung Lee. “Using Least Squares Support Vector Machines for Adaptive Communication Channel Equalization.” In: *International Journal of Applied Science and Engineering* 3 (1 Apr. 2005), pp. 51–59 (cit. on p. 3).

[Lim+24] Heimrih Lim Meng Kee et al. “A Review on Machine Learning for Channel Coding.” In: *IEEE Access* 12 (2024), pp. 89002–89025 (cit. on p. 2).

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on pp. 64, 85).

[MLL19] Manuel Eugenio Morocho-Cayamcela, Haeyoung Lee, and Wansu Lim. “Machine Learning for 5G/B5G Mobile and Wireless Communications: Potential, Limitations, and Future Directions.” In: *IEEE Access* 7 (2019), pp. 137184–137206 (cit. on p. 1).

[Moo05] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley, May 2005 (cit. on p. 12).

[Nac+17] Eliya Nachmani et al. *RNN Decoding of Linear Block Codes*. 2017 (cit. on p. 5).

[Nac+18] Eliya Nachmani et al. “Deep Learning Methods for Improved Decoding of Linear Codes.” In: *IEEE Journal of Selected Topics in Signal Processing* 12.1 (Feb. 2018), pp. 119–131 (cit. on pp. 5, 71, 72).

[NBB16] Eliya Nachmani, Yair Be’ery, and David Burshtein. “Learning to Decode Linear Codes Using Deep Learning.” In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, Sept. 2016 (cit. on pp. 4, 5, 87, 92).

[Niu+21] Kai Niu et al. “Deep Learning Methods for Channel Decoding: A Brief Tutorial.” In: *2021 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, July 2021 (cit. on p. 2).

[NW21] Eliya Nachmani and Lior Wolf. *Autoregressive Belief Propagation for Decoding Block Codes*. 2021 (cit. on pp. 5, 87).

[Par+23] Seong-Joon Park et al. *How to Mask in Error Correction Code Transformer: Systematic and Double Masking*. 2023 (cit. on pp. 6, 46, 79).

[Pfi17] Henry D. Pfister. “A Brief Introduction to Polar Codes.” In: 2017 (cit. on pp. 16, 24).

[Ram+09] R. Ramanathan et al. “Generalised and Channel Independent SVM Based Robust Decoders for Wireless Applications.” In: *2009 International Conference on Advances in Recent Technologies in Communication and Computing*. IEEE, Oct. 2009 (cit. on pp. 4, 28, 34).

[RL09] William Ryan and Shu Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, Sept. 2009 (cit. on pp. 4, 13, 17).

[Ros58] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408 (cit. on p. 4).

[Row+24] Mohammad Rowshan et al. “Channel Coding Toward 6G: Technical Overview and Outlook.” In: *IEEE Open Journal of the Communications Society* 5 (2024), pp. 2585–2685 (cit. on p. 1).

[RU01] T.J. Richardson and R.L. Urbanke. “The capacity of low-density parity-check codes under message-passing decoding.” In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 599–618 (cit. on pp. 45, 47, 48).

[Sal+14] S. Salcedo-Sanz et al. “Support vector machines in engineering: an overview.” In: *WIREs Data Mining and Knowledge Discovery* 4.3 (Apr. 2014), pp. 234–267 (cit. on p. 3).

[Sha48] C. E. Shannon. “A Mathematical Theory of Communication.” In: *Bell System Technical Journal* 27.3 (July 1948), pp. 379–423 (cit. on pp. 6, 60).

[She+20] Changyang She et al. “Deep Learning for Ultra-Reliable and Low-Latency Communications in 6G Networks.” In: *IEEE Network* 34.5 (Sept. 2020), pp. 219–225 (cit. on p. 1).

[Sim18] Osvaldo Simeone. “A Very Brief Introduction to Machine Learning With Applications to Communication Systems.” In: *IEEE Transactions on Cognitive Communications and Networking* 4.4 (Dec. 2018), pp. 648–664 (cit. on p. 1).

[SW20] Victor Garcia Satorras and Max Welling. *Neural Enhanced Belief Propagation on Factor Graphs.* 2020 (cit. on p. 5).

[SY16] V. Sudharsan and B. Yamuna. “Support Vector Machine Based Decoding Algorithm for BCH Codes.” In: *Journal of telecommunications and information technology* 2016 (2016) (cit. on pp. 3, 4, 28, 33, 34, 36, 40).

[Tek+19] Kursat Tekbiyik et al. “Multi–Dimensional Wireless Signal Identification Based on Support Vector Machines.” In: *IEEE Access* 7 (2019), pp. 138890–138903 (cit. on p. 3).

[TV11] Ido Tal and Alexander Vardy. “List Decoding of Polar Codes.” In: *2011 IEEE International Symposium on Information Theory Proceedings.* IEEE, July 2011 (cit. on pp. 24, 85).

[TV13] Ido Tal and Alexander Vardy. “How to Construct Polar Codes.” In: *IEEE Transactions on Information Theory* 59.10 (Oct. 2013), pp. 6562–6582 (cit. on p. 16).

[Vap97] Vladimir N. Vapnik. “The Support Vector method.” In: *Artificial Neural Networks — ICANN’97.* Springer Berlin Heidelberg, 1997, pp. 261–271 (cit. on pp. 3, 28).

[Vas+17] Ashish Vaswani et al. “Attention is All you Need.” In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017 (cit. on pp. 6, 52, 69).

[Vaz+21] Miguel Angel Vazquez et al. "Machine Learning for Satellite Communications Operations." In: *IEEE Communications Magazine* 59.2 (Feb. 2021), pp. 22–27 (cit. on p. 1).

[WW96] Xiao-An Wang and S.B. Wicker. "An Artificial Neural Net Viterbi Decoder." In: *IEEE Transactions on Communications* 44.2 (1996), pp. 165–171 (cit. on pp. 2, 4).

[Xu+17] Weihong Xu et al. "Improved Polar Decoder Based on Deep Learning." In: *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, Oct. 2017 (cit. on p. 5).

[YBW89] J. Yuan, V.K. Bhargava, and Q. Wang. "An Error Correcting Neural Network." In: *Conference Proceeding IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE, 1989 (cit. on p. 2).

[Ye+24] Neng Ye et al. "Artificial Intelligence for Wireless Physical-Layer Technologies (AI4PHY): A Comprehensive Survey." In: *IEEE Transactions on Cognitive Communications and Networking* (2024), pp. 1–1 (cit. on p. 1).

[ZHA89] G. Zeng, D. Hush, and N. Ahmed. "An Application of Neural Net in Decoding Error-Correcting Codes." In: *IEEE International Symposium on Circuits and Systems*. IEEE, 1989 (cit. on p. 2).

[Zib+16] Darko Zibar et al. "Machine Learning Techniques in Optical Communication." In: *Journal of Lightwave Technology* 34.6 (Mar. 2016), pp. 1442–1452 (cit. on p. 1).

[ZPH19] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. "Deep Learning in Mobile and Wireless Networking: A Survey." In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2224–2287 (cit. on p. 1).

Résumé — Dans cette thèse de doctorat, nous explorons des solutions basées sur l'apprentissage automatique pour le décodage canal dans les systèmes de communication de type Machine-à-Machine, où l'atteinte de communications ultra-fiables et à faible latence (URLLC) est essentielle. Leur principal problème provient de la croissance exponentielle de la complexité du décodeur à mesure que la taille du paquet augmente. Cette *malédiction de la complexité* se manifeste sous trois aspects différents : i) le nombre de schémas de bruit corrigeables, ii) l'espace des mots de code à explorer, et iii) le nombre de paramètres entraînables dans les modèles. Pour pallier la première limitation, nous explorons des solutions basées sur les Machines à Vecteurs de Support (SVM) et proposons une approche bit-à-bit qui réduit considérablement la complexité des solutions SVM existantes. Pour aborder la deuxième limitation, nous investiguons des décodeurs neuronaux de type *syndrome-based* et introduisons un nouveau décodeur orienté message qui améliore les schémas existants tant au niveau de son architecture que du choix de la matrice de parité. Concernant la taille du réseau, nous développons une version récurrente d'un décodeur basé sur le transformer qui réduit le nombre de paramètres tout en maintenant l'efficacité par rapport aux solutions précédentes. Enfin, nous étendons le décodeur proposé pour supporter les modulations d'ordre supérieur via les Modulations Codées avec et sans Entrelacement de Bits (BICM et CM, resp.), facilitant ainsi son application dans des environnements de communication plus réalistes.

Mots clés : Communications, codage canal, apprentissage automatique, réseaux de neurones, machine à vecteurs de support.

Abstract — In this Ph.D. thesis, we explore machine learning-based solutions for channel decoding in Machine-to-Machine type communications, where achieving ultra-reliable low-latency communications (URLLC) is essential. Their primary issue arises from the exponential growth in the decoder's complexity as the packet size increases. This *curse of dimensionality* manifests itself in three different aspects: i) the number of correctable noise patterns, ii) the codeword space to be explored, and iii) the number of trainable parameters in the models. To address the first limitation, we explore solutions based on a Support Vector Machine (SVM) framework and suggest a bitwise SVM approach that significantly reduces the complexity of existing SVM-based solutions. To tackle the second limitation, we investigate syndrome-based neural decoders and introduce a novel message-oriented decoder, which improves on existing schemes both in the decoder architecture and in the choice of the parity check matrix. Regarding the neural network size, we develop a recurrent version of a transformer-based decoder, which reduces the number of parameters while maintaining efficiency, compared to previous neural-based solutions. Lastly, we extend the proposed decoder to support higher-order modulations through Bit-Interleaved and generic Coded Modulations (BICM and CM, respectively), aiding its application in more realistic communication environments.

Keywords: Communications, channel coding, machine learning, neural networks, support vector machine.