DEEP LEARNING BASED DECODERS FOR CONCATENATED CODES OVER INSERTION AND DELETION CHANNELS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Eksal Uras Kargı

January 2025

Deep Learning Based Decoders for Concatenated Codes over Insertion and Deletion Channels

By Eksal Uras Kargı

January 2025

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Tolga M. Duman (Advisor)

Aykut Koç

Ahmed Hareedy

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

DEEP LEARNING BASED DECODERS FOR CONCATENATED CODES OVER INSERTION AND DELETION CHANNELS

Eksal Uras Kargı
M.S. in Electrical and Electronics Engineering
Advisor: Tolga M. Duman
January 2025

Channels with synchronization errors, including insertion/deletion channels, are of significant importance, as they are encountered in various systems, such as communication networks and various storage technologies, including DNA data storage. Serially concatenated codes where the outer code is a powerful channel code, such as a low-density parity-check (LDPC) or convolutional code, and the inner code is a watermark or marker code, are shown to be effective solutions over such channels. In particular, the use of marker codes, referring to insertion of preselected sequences in the transmitted data stream periodically, are shown to work well in regaining synchronization at the receiver and achieving improved error rate performance compared to other alternatives. In the current literature, maximum a posteriori (MAP) detector realized by the well-known forward-backward algorithm is commonly employed to decode the inner marker code and estimate the log-likelihood ratios (LLRs) of the bits encoded by the outer code, and the resulting log-likelihood estimates are fed to the outer decoder to estimate the transmitted data.

Alternative to the MAP detector, this thesis proposes deep learning-based solutions to estimate the LLRs of the coded bits in the paradigm of concatenated codes, exploiting the marker information and addressing some limitations of conventional methods. Bit-level deep learning-based detectors offer good alternatives when the channel statistics are not perfectly available at the decoder, degrading of the performance of the MAP detector. They can also be employed for one-shot decoding when the outer code is a convolutional code. Also developed are symbol-level deep learning-based detectors to exploit the correlations among adjacent bits

at the detector output. Contrary to the existing symbol-level decoders for insertion/deletion channels, the newly proposed approaches can go beyond the case of combining three bits, offering further enhancements in performance while keeping the complexity tolerable. As a final contribution, deep learning-based detectors are developed for insertion and deletion channels that are further exacerbated by inter-symbol interference, e.g., modeling bit-patterned media recording channels, and their performance is studied via numerical examples.

Keywords: Deep learning, deletion channel, insertion channel, channels with synchronization errors, concatenated codes, marker codes, recurrent neural networks, gated recurrent units, intersymbol inference.

ÖZET

EKLEME VE SİLME KANALLARINDA BİRLEŞTİRİLMİŞ KODLAR İÇİN DERİN ÖĞRENME TABANLI KOD ÇÖZÜCÜLER

Eksal Uras Kargı
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Tolga M. Duman
Ocak 2025

Senkronizasyon hatalı kanallar, ekleme/silme kanalları da dahil olmak üzere, iletişim ağları ve DNA veri depolama gibi çeşitli sistemlerde karşılaşıldığından dolayı önemli bir konudur. Dış kod olarak güçlü bir kanal kodu, örneğin düşük yoğunluklu parite denetim (low-density parity-check - LDPC) veya konvolüsyonel kod, ve iç kod olarak bir filigram veya işaret kodu kullanılan seri olarak birleştirilmiş kodlar, bu tür kanallar üzerinde etkili çözümler olarak gösterilmiştir. Özellikle, işaret kodlarının kullanımı, belirli dizilerin periyodik olarak iletilen veri akışına eklenmesi anlamına gelir ve alıcıda senkronizasyonu yeniden kazanmak ve diğer alternatiflere kıyasla daha iyi bir hata oranı performansı elde etmek için etkin bir yöntem olarak öne çıkmaktadır. Mevcut literatürde, işaret kodunu çözmek ve dış kod tarafından kodlanan bitlerin log-olasılık oranlarını (log likelihood ratio - LLR) tahmin etmek için iyi bilinen ileri-geri algoritma tarafından gerçekleştirilen maksimum sonsal olasılık (maximum a posteriori - MAP) dedektör yaygın olarak kullanılmakta ve elde edilen log-olasılık tahminleri dış kod çözücüsüne beslenerek iletilen veriyi tahmin etmektedir.

MAP dedektörüne alternatif olarak, bu tezde derin öğrenme tabanlı çözümler önerilmektedir; bu çözümler, birleştirilmiş kodlar paradigmasında kodlanmış bitlerin LLR'lerini tahmin etmek için işaret bilgilerini kullanmakta ve geleneksel yöntemlerin bazı sınırlamalarını çözmeye çalışmaktadır. Bit seviyesindeki derin öğrenme tabanlı dedektörler, kanal istatistiklerinin çözücüde tam olarak mevcut olmadığı durumlarda MAP dedektörünün performansını düşürdüğü zaman iyi alternatifler sunar. Ayrıca, dış kod bir konvolüsyonel kod olduğunda tek seferlik çözümleme için de kullanılabilirler. Bunun yanı sıra, dedektör

çıkışında bitler arasındaki korelasyonları kullanmak için sembol seviyesindeki derin öğrenme tabanlı dedektörler geliştirilmiştir. Ekleme/silme kanalları için mevcut sembol seviyesindeki kod çözücülerle zıt olarak, önerilen yeni yaklaşımlar, üç bitin birleştirilmesi durumunun ötesine geçerek performansta daha fazla iyileştirmeler sunabilir ve aynı zamanda kompleksiteyi tolere edilebilir seviyede tutar. Son olarak, bit desenli medya kaydı kanalları gibi durumları modelleyen semboller arası girişimle kötüleşen ekleme/slinme kanalları için derin öğrenme tabanlı dedektörler geliştirilmiş ve performansları sayısal örneklerle incelenmiştir.

Anahtar sözcükler: Derin öğrenme, silme kanalı, ekleme kanalı, senkronizasyon hataları olan kanallar, birleştirilmiş kodlar, işaret kodları, tekrarlayan sinir ağları, kapılı tekrarlayan birimler, semboller arası girişim.

Acknowledgement

I would like to begin by expressing my gratitude to Prof. Tolga M. Duman for his unwavering support over the years for my thesis.

I would like to express my heartfelt gratitude to my family for their unconditional, all-encompassing love and support throughout my graduate studies and all my endeavors. If I am where I am today, it is because of them.

I would like to express my gratitude to the members of the current and former CTAR Lab members whom I have had the privilege to know and befriend over the years: Furkan Bağcı, Muhammad Atif Ali, Büşra Tegin, Mert Özateş, Reza Asvadi, Mohammad Javad Ahmadi, Ruslan Morozov, Javad Haghighat, Ozan Aygün, Mohsen Moradi, Muhammed Kazemi, Mücahit Gümüş and Berke Eren. Each of these individuals has enriched my experience, contributing their knowledge, expertise, and friendship. The sense of community within the lab has transformed our collaborations into lasting friendships, deepening our mutual respect and making our journey more valuable over time.

I would like to thank my girlfriend, Esra Türeyen, for her unconditional support over the years.

I would also like to thank our faculty cat, Sultan, for her cute support during her visits to our room over the past three years, and my own cat, Charlie, whose purrs and occasional meows reminded me that even the toughest problems deserve a moment of feline wisdom and calm.

I feel lucky and valued for Vodafone's financial support of my thesis within the framework of the 5G and Beyond Joint Graduate Support Program, which is coordinated by the Information and Communication Technologies Authority. This thesis was also funded by the European Union through the ERC Advanced Grant 101054904: TRANCIDS. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or

the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Contents

List of Figures

List of Tables

List of Abbreviations

AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Rate
BI	Bi-directional
BSC	Binary Symmetric Channel
CSI	Channel State Information
CN	Check Node
DNA	Deoxyribonucleic Acid
FER	Frame Error Rate
GRU	Gated Recurrent Unit
HDD	Hard-Decision Decoding
i.i.d.	independent and identically distributed
ISI	Intersymbol Interference
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood Ratio
LSTM	Long Short-Term Memory
MAP	Maximum A Posteriori
MLP	Multi Layer Perceptron
RNN	Recurrent Neural Network
RL	Reinforcement Learning
SNR	Signal-to-Noise Ratio
SDD	Soft-Decision Decoding
VN	Variable Node
VT	Varshamov-Tenengolts

Chapter 1

Introduction

1.1 Overview

This thesis proposes deep learning-based decoders for concatenated codes over channels with synchronization errors. Channels with synchronization errors, also known as insertion and deletion channels, are observed in various real-world, practical applications. These channels either delete bits from the data stream or add random bits to it. Such random deletions or insertions create synchronization problems for the receiver to resolve. DNA data storage, one of the emerging, nascent technologies to offer solutions for digital storage, is an exemplary application where such errors are observed. Random deletions or insertions coupled with technologies' limitations in storing information in very long strands and the requirement to store DNA in a liquid consisting of scores of strands without order information make DNA data storage highly challenging. As another example, bit patterned media recording channels for digital storage suffer from synchronization errors. Synchronization errors are also observed in communication systems where the receiver and transmitter clocks are mismatched.

Motivated by the above technologies and the challenges they face, different types of mathematical models for channels with synchronization errors have been

developed. For example, nanopore channels are nanoscale pores embedded in membranes that allow the passage of DNA strands. They suffer not only from synchronization errors but also from inter-symbol inference (ISI), as the current stemming from the passage of the strand is affected by previous nucleotides. Another example where ISI is observed along with insertions and deletions is bit patterned media recording systems, which is a magnetic storage technology where the sequence of bits are mapped onto bit islands.

The capacity of channels with independent and identically distributed (i.i.d.) insertions, deletions, and substitutions, have been examined in literature. It was proved that such channels are information stable; hence, their Shannon capacity exists. However, the determination of the capacity expression has remained elusive, while various upper and lower bounds have been developed. These bounds on the channel capacity serve as benchmarks for comparing the performance of practical channel codes with ultimate limits of communication.

Practical codes for insertion and deletion channels have also been developed. These can be categorized into three types: short block codes, convolutional codes, and concatenated codes. An important block code example is Varshamov-Tenengolts (VT) codes which are capable of correcting a limited number of deletions and insertions employing suitable syndromes. Convolutional codes serve as alternatives and they could be used with different trellis based decoding algorithms. Another early realization in the field was the use of specific sequences of binary strings inserted into the codeword sequences to allow the receiver to detect these sequences and recover the original data. An effective solution for coding over insertion/deletion channels is the use of concatenated codes. For instance, combination of low-density parity-check (LDPC) codes and codes obtained by specific sequences, such as marker codes or watermark codes which are known to both the receiver and transmitter, offer good solutions. Fig. 1.1 shows the block diagram of a concatenated coding approach where the inner code is a marker code.

More specifically, concatenated codes can be categorized into two types: those employing watermark sequences and those employing marker sequences. These

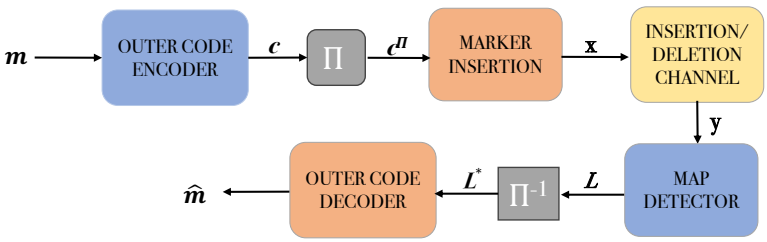

Figure 1.1: Block diagram of concatenated codes.

two are used as inner codes for concatenated codes, with, for instance, LDPC codes as outer codes. Marker codes are sequences of bits known to both the receiver and the transmitter and are inserted regularly into the encoded sequences of the outer codes. Similarly, watermark sequences are added componentwise to the encoded sequence. Both types of sequences allow the receiver to detect synchronization errors by comparing the received sequence through the channel with the known sequence of the watermark or marker sequence, aiming for the same goal. The receiver uses this information to regain synchronization. Concatenated codes are decoded by using maximum a posteriori (MAP) detection algorithm that utilizes information from the known sequences of markers or watermarks. The MAP detection algorithm estimates the posterior probabilities of the bits encoded by the outer codes, and the outer code uses this information to achieve competitive error-correcting performance. Although both methods aim for the same goal, marker codes outperform watermark codes for communication over insertion/deletion channels. Therefore, our focus in this thesis is on the use of marker codes as inner codes.

The main method for regaining synchronization in the setup of concatenated codes, the MAP detection algorithm, employs the forward-backward algorithm also known as the BCJR algorithm. This algorithm requires detailed calculations and channel statistics to function properly; hence, it underperforms when channel parameters are not fully known. Furthermore, the performance of the MAP detector improves when MAP detection is implemented at the symbol level (considering multiple bits together) rather than the bit level. Symbol-level BCJR decoders perform better than bit-level ones because they exploit the correlations among the bits input to the channel. While derivations for the bit-level case are

easier to obtain, the symbol-level derivations are complicated, especially when the number of bits comprising each symbol exceeds two or three, and their implementation complexity increases drastically.

Deep learning techniques for channel coding has been a recently developing area. The research in this field started with applying basic neural networks, such as Hopfield networks, to decode simple codes with small codeword lengths. Later, multi-layer perceptions (MLPs) with numerous hidden layers were adopted for channel codes, and it was shown that it is possible to achieve MAP detection performance for small codeword lengths. This has evolved into decoding convolutional codes with sequential neural network architectures, especially after the advent of new architectures such as recurrent neural networks. Nowadays, transformers which represent one of the landmark achievements of deep learning for language and computer vision tasks are being applied to decode channel codes. Neural network-based decoders aim to achieve low-latency and high-performance decoding capabilities. Despite these advancements, decoding with neural networks becomes intractable as codeword lengths increase. Note also that neural networks are employed not only for decoding but also for constructing state-of-the-art codes, such as polar codes, through reinforcement learning.

In this thesis, deep learning based architectures are developed to regain synchronization at the receiver for channels with synchronization errors. By replacing the MAP detector module, deep learning models estimate the log likelihood ratios (LLRs) of the bits coded by the outer code, and offering an alternative solution to the current approaches.

1.2 Thesis Contributions

With the overall goal of developing low complexity and robust solutions for receiver design for insertion/deletion channels, this thesis utilizes sequential deep learning architectures to estimate probabilities of the sequences encoded by outer

codes within the framework of concatenated codes. Two different codes are considered as outer codes: LDPC codes with different lengths and convolutional codes. In addition, two different setups are considered: in the first setup, we use a serial concatenation of an LDPC code (or, a convolutional code) with a marker code; a sequential deep learning architecture is developed to estimate the LLRs of the bits input to the channel. In the second setup, we focus on the serial concatenation of a convolutional code with a marker code and develop deep learning architectures that perform one-shot decoding. In this case, two architectures are combined to estimate the LLRs and decode the message bits in a combined bi-directional gated recurrent unit (BI-GRU) based receiver. Our results show that the resulting error rate performance is competitive with those of the approaches using the BCJR algorithm-based receivers for decoding the inner code.

BCJR-based solution requires the exact channel state information for estimating the LLRs optimally. To address this issue, we adopt deep learning-based decoders that do not require the exact channel statistics. We show that using a deep learning decoder may improve performance in real-world scenarios where channel parameters is unknown. Our results along these lines of investigations are published in [1] and presented at the 2024 IEEE International Conference on Communications (ICC).

Symbol-level deep learning-based decoding, which improves upon the bit-level one, is also developed to utilize deep learning techniques' potential in this setup. Symbol-level decoding is an improvement by exploiting the correlations among the bits, neglected in the bit-level decoders. In the literature, symbol-level MAP detectors are available, however, they are highly complicated and hence, they are only limited to grouping only two or three bits, offering modest performance improvements.

Our third contribution is to develop deep learning-based decoders for decoding concatenated codes over insertion and deletion channels with ISI. Symbol-level deep learning decoders are employed to address both synchronization errors and the effects of ISI. The conventional methods have the drawback that the number of required states in the overall trellis is extremely high, increasing the

complexity, hence making their implementation impractical. Numerical experiments indicate that deep learning methods for decoding concatenated codes over insertion/deletion channels with ISI offer promising solutions that address the complexity issue.

1.3 Thesis Organization

The thesis is organized as follows. Chapter 2 provides a thorough literature review on channels with synchronization errors, including computable upper and lower bounds on their capacity, concatenated codes for the channels of interest, detection algorithms in the concatenated code setting, and application of neural networks to decode channel codes for different channel models.

Chapter 3 describes the proposed solution for communication over insertion/deletion channels using bit-level deep learning architectures. It describes novel deep learning-based estimators, presents extensive numerical examples demonstrating their error rate performance, and gives achievable rates with the proposed setup over insertion/deletion channels.

Chapter 4 proposes symbol-level deep learning based decoders, as an improvement over the bit-level approach of Chapter 3. The newly developed decoders offer solutions to scenarios when conventional symbol-level detectors are not suitable. Extensive numerical examples demonstrate the error rate performance of the proposed architectures and compare the results with the baseline symbol-level detection algorithms.

Chapter 5 proposes an extension of the symbol-level deep learning decoders presented in Chapter 4 to the case of insertion and deletion channels with ISI. The foremost aim is to show that deep learning methods could be utilized not only in basic insertion and deletion channels but also for other more complicated channel models. ISI worsens the channel conditions and the conventional approaches become highly complicated when there are also synchronization errors, motivating

the use of deep learning based solutions. Some numerical results on the system performance are also presented.

Finally, Chapter 6 provides conclusions and future directions for research on coding over insertion and deletion channels, and applications of deep learning techniques for communicating over these channels.

Chapter 2

Preliminaries and Literature Review

This chapter aims to familiarize the reader with insertion and deletion channels. It also explores the motivation behind choosing concatenated codes as suitable channel codes over these channels. A detailed discussion on maximum a posteriori (MAP) detection algorithms for concatenated codes is provided, followed by a comparison with deep learning-based architectures proposed in subsequent chapters. This comparison gives insights into the similarities between traditional MAP-based methods and deep learning decoders. Additionally, deep learning-based decoders for various channel codes over different channels are examined, providing a perspective on why deep learning architectures could be considered promising candidates.

The chapter begins by presenting mathematical models for insertion and deletion channels, reviews available computable lower and upper bounds on the capacity of channels with independent and identically distributed (i.i.d.) synchronization errors, and presents the different channel codes over insertion/deletion channels in Section 2.1. Then, the details of the outer code decoders, the sum-product algorithm for low-density parity-check (LDPC) codes, and the Viterbi

decoder for convolutional codes, are given in Section 2.2. Following this, concatenated codes are introduced for the channels of interest in Section 2.3. The MAP detection algorithm, implemented using the forward-backward algorithm or the BCJR algorithm, is explained in detail, including derivation of the forward and backward recursions. Finally, deep learning-based models for other channels are discussed to motivate their potential use for decoding concatenated codes over insertion and deletion channels in Section 2.4. The chapter is concluded in Section 2.5.

2.1 Channels with Synchronization Errors

Insertion and deletion channels or channels with synchronization errors, are encountered in various real-world scenarios, such as storage systems, recording channels, and wireless communications [2, 3, 4]. In the wireless communications setup, the clocks of the transmitter and receiver ends of the communication system can be mismatched due to problems inside the system, especially in the clock, resulting in discrepancy in synchronization, resulting in bit deletions and insertions.

For conventional recording systems, achieving ultra-high recording densities is impeded by the nature of electromagnetic material embedded into these systems [4]. In order to overcome the hindrances of the nature of the materials in conventional media recording systems and to achieve high recording densities, bit-media patterned recording systems are proposed by prepatterning the media into small chunks, known as bit islands [5]. The overarching goal of these systems is to provide high recording densities. The islands on which bits are recorded have imperfections due to the nature of fabrication, resulting in the deletion and insertion of bits during the storing process. These errors can be considered as synchronization errors.

Another scenario in which insertion/deletion channel models are observed is DNA data storage, where existing nucleotides are deleted, or new ones are inserted into the DNA strands [6]. DNA data storage is a technology promising to

store high amounts of data by mapping information bits into DNA strands. This technology also offers high longevity. DNA size being on the scale of nanometers and a strand of DNA offering storage on the scale of terabytes make them promising storage options [7].

Although DNA data storage is a promising technology thanks to its capability of storing a high number of bits on the scale of nanometers and high longevity, it has some drawbacks. For instance, due to technical limitations, information is stored in many short DNA strands, rather than one long strand. Another limitation is that DNA strands should be stored inside a solution, called the DNA pool [8]. The DNA pool contains myriad of strands since information is stored on different DNA strands, and the order of the strands is lost. Moreover, DNA strands abound in the DNA pool since they are amplified via the polymerase chain reaction process. During the retrieval process of the stored information, random sampling of DNA strands is required to fetch the DNA strands. This process of random sampling from a large number of strands is modeled mathematically by a noisy sampling channel and its variants [9]. These channels experience errors not only during the sampling and ordering process but also due to the random insertion and deletion of nucleotides in DNA strands.

2.1.1 Insertion and Deletion Channel Models

There are various models for insertion and deletion channels. The first to be reviewed is the Mackay channel model [10]. In this model, probabilities P_d, P_t, and P_s govern the binary channel. At time step t, a bit can encounter three different events for each channel use. Bit at timestep t, enters the queue awaiting to be transmitted. With probability P_i, a random bit is added to the sequence, and it enters the same process. The number of inserted bits is a geometrically distributed random variable with parameter P_i. With probability P_d, the bit at position t is deleted, with probability $1 - P_d - P_i$, the bit at position t is transmitted, being susceptible to substitution error with probability P_s. The state diagram of this channel at time step t is given in Fig. 2.1.

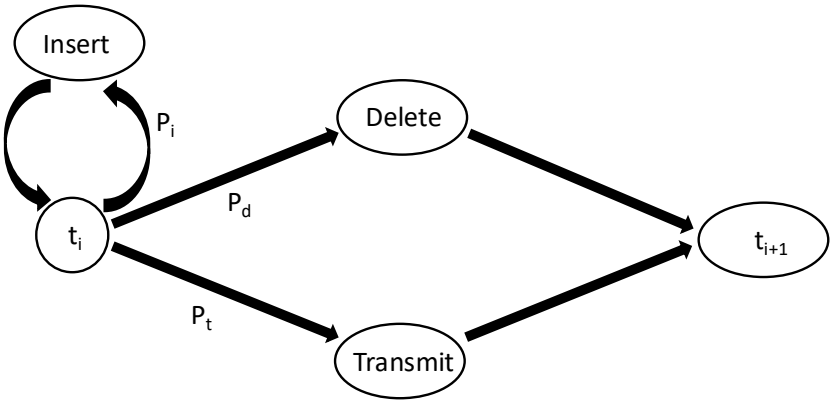

Figure 2.1: Mackay insertion/deletion channel model.

Another widely used model for channels with synchronization errors puts a limitation on a number of inserted bits if an insertion event occurs. In this model, referred as Gallager model [11], two uniformly random bits are inserted into the sequence if an insertion event occurs (which has probability P_i). Deletion and transmission events occur similarly to the Mackay's model, with probabilities P_d and P_t, respectively. There are many other insertion/deletion channel models. For instance, for a sticky channel model, where each symbol may be replicated multiple times following a certain probability mass function.

Some channel models encounter intersymbol interference (ISI) along with insertions and deletions. Authors in [12] model such channels to account for synchronization errors and ISI in bit-patterned media recording systems. This channel is modeled as a cascade of insertion and deletion channel followed by an ISI channel. Another important insertion and deletion channel model with ISI appears the nanopore sequencing channel model [13]. Nanopore sequencing is a technology that synthesizes digital data into DNA strands by passing the sequence through a biological membrane of nanometer size. As nucleotides pass through the membrane, they generate current levels from which the sequencer must determine which nucleotide has passed. However, these current levels are affected not only by the nucleotides currently passing through the channel but also by previous nucleotides, leading to inter-symbol interference from earlier bits.

2.1.2 Capacity Results on Insertion and Deletion Channels

In this section, capacity and computable capacity upper and lower bounds for channels with synchronization errors are discussed. Dobrushin proved that channels with i.i.d. synchronization errors are information stable; hence, their Shannon capacity exists [14]. The channel capacity is given as,

$$C = \lim_{N\to\infty} \max_{P(X)} \frac{1}{N} I(X;Y) \tag{2.1}$$

where X is the channel input of length N, Y is the channel output, $P(X)$ is the distribution of the input sequence, and $I(\cdot;\cdot)$ denotes the mutual information.

While the channel capacity exists and it can be written as shown in (2.1), it does not seem possible to compute it for many insertion/deletion channel models. On the other hand, computable upper and lower bounds exist. A readily available upper bound on the capacity of i.i.d. deletion channel model is obtained by a direct comparison between the erasure and deletion channels. The capacity of an i.i.d. deletion channel is lower than that of an erasure channel, because, unlike the erasure channel where the erasure positions are known to the receiver, the deletion positions in a deletion channel are unknown [15]. With the deletion pattern as side information, deletion channel is converted to an erasure channel. Other upper bounds can also be derived by providing the receiver with side information, thereby converting the channel into simpler channels, whose capacities can be computed [16, 17, 18, 19, 20].

The first lower bound on the capacity of deletion channel was derived in [11]. In this paper, Gallager proved that for deletion probabilities lower than 0.5, i.e., $P_d \leq 0.5$, the capacity of the i.i.d. deletion channel is at least equal to capacity of binary symmetric channel (BSC) with cross-over probability P_d. Another lower bound on capacity of i.i.d. insertion and deletion channels is found in [21] by focusing on the transmission capacity, and evaluating the error probability via combinatorial arguments similar to Shannon capacity results. In [22], authors found lower bounds on deletion and duplication channels.

2.1.3 Channel Codes for Insertion and Deletion Channels

Most channel codes are developed for synchronous channels, meaning that they can account and correct for substitution errors, for which some symbols of the original sequence are replaced by other symbols with some probability. For channels with synchronization errors, a different approach is needed to construct codes that can provide error correction. Different approaches have been developed as practical channel codes over such channels, including algebraic and number theoretic codes, convolutional codes, and concatenated codes with marker and watermark codes.

Levenstein discovered that certain algebraic code constructions can be adopted for correcting deletion errors [23]. Reference [24] gives an introductory exposition to insertion and deletion error correcting codes. Varshamov-Tenengolts (VT) codes are nearly optimal for binary-input deletion channel [25] and can correct an arbitrary single deletion or single substitution error. [26] provides new insights into the structure of VT codes and their variants.

Convolutional codes are also employed for coding over insertion and deletion channels. By introducing new states into the Viterbi decoder and leveraging stack decoder with some changes, various decoders for convolutional codes are proposed [27]. Another decoder for trellis-based codes are proposed in [28]. Recently, a modified Fano decoder was developed for decoding of convolutional codes over insertion, deletion channels.

From early on, it has been recognized that an effective approach for communication over insertion and deletion channels is to employ pre-selected markers, known both to the transmitter and receiver, into the information sequence that will be sent through the channel [29]. An important class of codes for insertion/deletion channels is obtained by concatenating an outer code with an inner marker or watermark code [10, 30, 31]. This approach incorporates a powerful channel code as an outer code to correct for errors and marker or watermark codes as inner codes, which help regain synchronization. Markers are specific sequences of bits inserted into the data stream at regular intervals. They serve as

reference points for maintaining synchronization, allowing the receiver to detect if there are insertions or deletions among them. The outer code can be a powerful channel code such as an LDPC code [30, 31].

As an alternative approach, watermark codes employ a predetermined sequence of bits, similar to markers, added component-wise to a sparsely encoded sequence of bits to recognize and recover from insertion and deletion errors. They can be efficiently used with non-binary LDPC codes [10]. Watermark resembles a carrier signal and its main goal is to detect irregularities and discontinuities in this received signal by comparing received sequence to the watermark sequence.

MAP detection is used to identify the synchronization errors by estimating the log-likelihood ratios (LLRs) of the transmitted bits, which are then input to the outer decoder for serial concatenation of an outer code with an inner code over channels with insertions and deletions [10]. MAP detection is implemented in the bit-level, meaning that LLRs are calculated for each bit assuming they are uncorrelated. This results in a loss of information. Hence, an enhanced version, called symbol-level decoding, considering multiple bits together [31], is also proposed. While symbol-level decoding offers performance improvements, it is more complex than bit-level decoding, and it becomes infeasible for symbol lengths beyond a few bits.

Recently, deep learning based decoders are also developed for communication over insertion/deletion channels [32]. The authors propose two distinct setups: a model-based configuration within the framework of forward-backward equations and a model-free setup, where the implementation focuses on end-to-end estimation of LLRs within a framework akin to our development in Chapter 3. The model-based setup aims to estimate the channel probabilities if the channel state information (CSI) is not known to the receiver and utilizes these probabilities in the forward-backward equations. While our work in Chapter 3 has similarities to this independent parallel work, there are also substantial differences as will be clarified.

2.2 Review LDPC and Convolutional Codes

2.2.1 LDPC Codes

Bit-level LDPC codes are defined by a sparse parity check matrix $\mathbf{H}$ with size $m \times n$, where n denotes the number of codeword bits, k denotes the number of message bits and $m \geq n - k$ denotes number of parity check equations. The rate is $r = k/n$. There are various LDPC code design methods for $\mathbf{H}$. After finding an appropriate generator matrix, $\mathbf{G}$ of size $k \times n$, that ensures $\mathbf{GH}^T = \mathbf{0}$ where $\mathbf{0}$ is all-zero matrix of 0's, $\mathbf{y} = \mathbf{Gx}$ gives an encoding where $\mathbf{x}$ is message vector and $\mathbf{y}$ is codeword vector [33, 34].

There are iterative methods for decoding of LDPC codes. The most popular is the iterative sum-product algorithm. There are k variable nodes (VNs) and m check nodes (CNs) for a given $\mathbf{H}$ matrix. CNs are code constraints defined by the rows of the $\mathbf{H}$ matrix and VNs are the codeword bits represented by its columns.

The sum-product algorithm takes the LLRs of the bits input to the channel, $\mathbf{L} = (L_1, \ldots, L_n)$. The LLR of the k^{th} bit, L_k, for $k \in \{1, \ldots, n\}$, is given by:

$$L_k = L(x_k \mid \mathbf{y}) = \frac{P\left(x_k = 0 \mid \mathbf{y}\right)}{P\left(x_k = 1 \mid \mathbf{y}\right)}.$$

An iteration of the decoding algorithm consists of sending information from VNs to CNs, then sending information from CNs to VNs. The information sent through the i^{th} VN node to the j^{th} CN node at iteration number t is denoted by $V_t(i \to j)$ and the information sent through the j^{th} CN node to the i^{th} VN node at iteration number t is denoted by $C_t(j \to i)$. The decoding algorithm starts with updating the VN nodes with channel LLRs:

$$V_1(i \to j) = \begin{cases} L_i & \text{if } H_{i,j} = 1, \\ 0 & \text{if } H_{i,j} = 0, \end{cases} \quad \forall i, j. \tag{2.2}$$

The CN to VN messages can be expressed as follows where $N_C(j)$ denotes the indexes with $H_{i,j} = 1$ for $i = 1, \ldots, m$ and $\tanh(x)$ is the hyperbolic tangent

function, defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$:

$$C_t\left(j \to i\right) = \begin{cases} 2\tanh^{-1}\left(\prod_{k \in N_C(j)\setminus\{i\}} \tanh\left(\frac{1}{2}V_t(k \to i)\right)\right) & \text{if } H_{i,j} = 1, \\ 0 & \text{if } H_{i,j} = 0, \end{cases} \quad \forall i, j. \tag{2.3}$$

After the first iteration, $t \in \mathbb{N}$, the i^{th} VN acts as an repetition code. VN to CN messages is calculated as follows where $N_V(i)$ denotes indexes with $H_{i,j} = 1$ for $j = 1, \ldots, n$:

$$V_{t+1}\left(i \to j\right) = \begin{cases} L_i + \sum_{k \in N_V(i)\setminus\{j\}} C_t(k \to i) & \text{if } H_{i,j} = 1, \\ 0 & \text{if } H_{i,j} = 0, \end{cases} \quad \forall i, j. \tag{2.4}$$

The LLR of the i^{th} bit at timestep t, $L_t(x_i)$, is updated as follows for $i \in 1, \ldots, n$:

$$L_t(x_i) = L_i + \sum_{k \in N_V(i)} C_{t-1}(k \to i) \tag{2.5}$$

The LLRs can also be converted to hard decisions values by comparing them with threshold 0, namely,:

$$\hat{x_i} = \begin{cases} 1 & \text{if } L_t(x_i) > 0 \\ 0 & \text{if } L_t(x_i) \leq 0 \end{cases} \quad \forall i. \tag{2.6}$$

The iterative decoding algorithm ends if the total number of pre-determined iterations, denoted by t_{max}, are exceeded or at any iteration, the algorithm yields a valid codeword, i.e $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$ where $\mathbf{0}$. There are other simplified, low-complexity decoding algorithms for LDPC codes. The foremost to mention is the min-sum algorithm [33] which updates CN to VN messages using a simplified approach. The min-sum decoder replaces $C_t\left(j \to i\right)$, and $V_t\left(i \to j\right)$ as follows,

$$C_t\left(j \to i\right) = \begin{cases} \prod_{k \in N_C(j)\setminus\{i\}} \alpha_{k,j} \min_{k \in N_C(j)\setminus\{i\}} \beta_{k,j} & \text{if } H_{i,j} = 1, \\ 0 & \text{if } H_{i,j} = 0, \end{cases} \quad \forall i, j. \tag{2.7}$$

Examples of other low-complexity decoders whose details are beyond the scope of this thesis, include the attenuated offset min-sum decoders, and the reduced-complexity box-plus decoder [33], among others.

2.2.2 Convolutional Codes

Convolutional codes form a different class of error-correcting codes, which generate the codeword sequences via convolution of binary vectors with the input data stream [33, 34]. Time-invariant convolutional codes are defined by r finite impulse response vectors, $\mathbf{h}_i = (h_{i,1}, \ldots, h_{i,t})$ where t denotes the number of delay elements and $h_{i,j} \in \{0, 1\}$ for $i = 1, \ldots, r$, and $j = 1, \ldots, t$. Given input vector $\mathbf{x} = (x_1, x_2, \ldots, x_k)$, encoding can be realized by taking convolutions of $\mathbf{x}$ and $\mathbf{h}_i$, and obtaining r different output streams. That is,

$$y_{i,j} = \sum_{m=1}^{t} h_{i,m} \cdot x_{j-m+1}, \tag{2.8}$$

where the operations are in the binary field. There are different representations of codes for convolutional codes, e.g., using state diagrams, code trellises.

Two main decoding algorithms for convolutional codes are the BCJR and Viterbi algorithms [35]. The BCJR algorithm minimizes the bit error probability, while the Viterbi algorithm minimizes the block error probability. In this thesis, the Viterbi decoder is used to decode convolutional codes. Let us briefly explain the idea. The maximum likelihood decision is given as follows,

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} P(\mathbf{y} \mid \mathbf{x}).$$

where $\mathbf{y} = (y_1, \ldots, y_n)$ is the output sequence of the channel of interest. Solving this problem can easily be shown to finding minimum Hamming difference between $\mathbf{y}$ and all possible codewords of $\mathbf{x}$ for BSC and finding minimum Euclidean distance for an additive white Gaussian noise (AWGN) channel. The Viterbi decoder utilizes this observation, and obtains the path through the trellis diagram for which the Hamming distance or the squared Euclidian distance is minimized, in a highly efficient manner. More details on the Viterbi decoder are available in [33, 34].

Message Bits

01011 01111 00101

LDPC /Convolutional Codeword Bits

00110 11010 01111 11011 00001 10110

Marker Insertion to Codeword Bits

00110 01 11010 01 01111 01 11011 01 00001 01 10110 01

Figure 2.2: Insertion of marker bits into the LDPC or convolutional coded bits. This case is for when $N_c = 5$, $N_m = 2$, $k/n = 0.5$, $r = 5/14$ and the two-bit marker of $[0, 1]$.

2.3 Concatenated Codes for Insertion/Deletion Channels

In this section, the system model for concatenated codes with marker codes as inner codes and the MAP detection realized by the BCJR algorithm are reviewed. Message vector $\mathbf{m}$ is encoded via the outer code with rate k/n, and then the encoded sequence $\mathbf{c}$ is interleaved in the bit-level. Then, a pre-selected marker sequence with N_m bits is inserted regularly into the coded bits. The resulting sequence $\mathbf{x}$ with length T is transmitted over insertion and deletion channel. The rate of inner code, marker code is, $r_m = N_m/(N_c + N_m)$. The rate of the overall code is $r = r_o r_m = \frac{N_c k}{(N_c+N_m)n}$. The resulting sequence of T bits comprise the overall codeword as illustrated in Fig. 2.2.

The received sequence is $\mathbf{y}$ with length R. At the receiver, a MAP detector can be used to estimate the LLRs of the bits encoded by the inner code. The LLRs can be rewritten using the Bayes rule as

$$L_k = L(x_k \mid \mathbf{y}_1^R) = \frac{P\left(\mathbf{y}_1^R \mid x_k = 0\right) P\left(x_k = 0\right)}{P\left(\mathbf{y}_1^R \mid x_k = 1\right) P\left(x_k = 1\right)}. \tag{2.9}$$

The algorithm utilizes the perfect a-priori information from the locations of marker, in other words, the receiver has the perfect information on whether the

k^{th} transmitted bit is a marker bit, therefore, it has the full knowledge of $P(x_k)$. If the k^{th} transmitted bit is not a marker bit, then $P(x_k) = 1/2$ for $x_k = \{0, 1\}$; hence, the LLR definition in (2.9) reduces to $L_k = \frac{P\left(y_1^R|x_k=0\right)}{P\left(y_1^R|x_k=1\right)}$, showing that recevier has no information about the bit. That LLRs of the bits corresponding to marker bits is trivial, meaning that they are either $+\infty$ if marker bit is 0 due to the $P\left(x_k = 0\right) = 1$ or $-\infty$ if marker bit is 1 since in this case $P\left(x_k = 1\right) = 1$. Therefore, the LLRs pertaining to marker bits are not important, and they are removed, i.e., only the LLRs of the bits encoded by the outer code remain. Then, these LLRs (belonging to the bits encoded by the outer code) are deinterleaved, and they are input to the outer code decoder. The output of the outer code decoder is the message bit estimates, $\hat{\mathbf{m}}$. The outer code's decoder can be selected as the sum-product decoder for an LDPC code, or the Viterbi decoder for a convolutional code, or any generalized bit-level decoder pertinent to the outer code.

The LLRs of the bits encoded by inner code are found by calculating the likelihoods, $P\left(\mathbf{y}_1^R \mid x_k = 0\right)$ and $P\left(\mathbf{y}_1^R \mid x_k = 1\right)$. Calculation of these terms is realized by the general forward backward algorithm, or known as the BCJR algorithm, that minimizes the bit-error probability [36]. The state governing the BCJR equations is defined differently for different setups: In [31], the state is defined as $D_{i,j}$ where i denotes the number of received bits in j transmitted bits, $i \in \{0, \ldots, T\}$, $j \in \{0, \ldots, R\}$ whereas in [10], the state $D_{i,j}$ denotes at stage j, the difference between transmitted and received bits is i, also known as the drift, $i \in \mathbb{N}$, $j \in \{0, \ldots, R\}$. The range of i is not limited, however, for practical reasons there is a limitation on the amount of drift. In order to implement the algorithm, references [10, 31] assume the number of received bits, R, is known to the receiver. This assumption is reasonable since current communications systems can determine frame boundaries. The forward and backward variables in [31] are defined as follows:

$$\alpha_i(j) \triangleq P\left(\mathbf{y}_1^j, D_{i,j}\right), \tag{2.10}$$

$$\beta_i(j) \triangleq P\left(\mathbf{y}_{j+1}^R \mid D_{i,j}\right). \tag{2.11}$$

We commence with derivation with the forward equation. These derivations use the Gallager insertion/deletion channel model explained in Section 2.1. We can

use the definition and decompose it by using the total probability theorem:

$$\begin{aligned}\alpha_i(j) &= P\left(\mathbf{y}_1^j, D_{i,j}\right) \\ &= P\left(\mathbf{y}_1^j, D_{i,j}, D_{i-1,j-2}\right) + P\left(\mathbf{y}_1^j, D_{i,j}, D_{i-1,j-1}\right) + P\left(\mathbf{y}_1^j, D_{i,j}, D_{i-1,j}\right).\end{aligned}$$

Similarly, the probability terms are simplified by conditioning and applying the definition of forward variable,

$$\begin{aligned}\alpha_i(j) &= P\left(\mathbf{y}_1^{j-2}, D_{i-1,j-2}\right) P\left(y_{j-1}^j, D_{i,j} \mid \mathbf{y}_1^{j-2}, D_{i-1,j-2}\right) \\ &\quad + P\left(\mathbf{y}_1^{j-1}, D_{i-1,j-1}\right) P\left(y_j, D_{i,j} \mid \mathbf{y}_1^{j-1}, D_{i-1,j-1}\right) \\ &\quad + P\left(\mathbf{y}_1^j, D_{i-1,j}\right) P\left(D_{i,j} \mid \mathbf{y}_1^j, D_{i-1,j}\right) \\ &= \alpha_{i-1}(j-2) P\left(y_{j-1}^j, D_{i,j} \mid D_{i-1,j-2}\right) + \alpha_{i-1}(j-1) P\left(y_j, D_{i,j} \mid D_{i-1,j-1}\right) \\ &\quad + \alpha_{i-1}(j) P\left(D_{i,j} \mid D_{i-1,j}\right).\end{aligned}$$

$P\left(D_{i,j} \mid D_{i-1,j}\right) = P_d$ since the transition denotes the deletion event and, $P\left(y_{j-1}^j, D_{i,j} \mid D_{i-1,j-2}\right) = P_t/4$ since the transition denotes the insertion event and two bits are randomly chosen from uniform according to the channel model. $P\left(y_j, D_{i,j} \mid D_{i-1,j-1}\right) = P_t G(x_i, y_j)$ where

$$G(x,y) = \begin{cases} 1 - P_s & \text{if } x = y, \\ P_s & \text{if } x \neq y, \end{cases}$$

since the transition denotes the transition event. Finally, the governing recursive forward equation for $i \in \{0, \ldots, T\}$ and $j \in \{0, \ldots, R\}$ is given as follows,

$$\alpha_i(j) = P_d \alpha_{i-1}(j) + P_t \alpha_{i-1}(j-1) \sum_{x_i} P(x_i) G(x_i, y_j) + \frac{P_i}{4} \alpha_{i-1}(j-2).$$

In order to start the recursion, initial conditions must be known. The initial conditions are as follows:

$$\alpha_0(j) = \begin{cases} 1, & \text{if } j = 0, \\ 0, & \text{if } j \in \{1, 2, \ldots, R\}. \end{cases}$$

Let us now write the backward recursion. We follow the similar steps with the forward recursion equations. We can decompose the backward variable by the total probability theorem:

$$\begin{aligned}\beta_i(j) &= P\left(\mathbf{y}_{j+1}^R \mid D_{i,j}\right) \\ &= P\left(\mathbf{y}_{j+1}^R, D_{i+1,j} \mid D_{i,j}\right) + P\left(\mathbf{y}_{j+1}^R, D_{i+1,j+1} \mid D_{i,j}\right) + P\left(\mathbf{y}_{j+1}^R, D_{i+1,j+2} \mid D_{i,j}\right).\end{aligned}$$

Similarly, probability terms are simplified, that is,

$$\begin{aligned}\beta_i(j) &= P\left(D_{i+1,j} \mid D_{i,j}\right) P\left(\mathbf{y}_{j+1}^{R} \mid D_{i+1,j}\right) \\ &\quad + P\left(D_{i+1,j+1} \mid D_{i,j}\right) P\left(\mathbf{y}_{j+2}^{R} \mid D_{i+1,j+1}\right) P\left(y_{j+1} \mid D_{i+1,j+1}, D_{i,j}\right) \\ &\quad + P\left(D_{i+1,j+2} \mid D_{i,j}\right) P\left(\mathbf{y}_{j+3}^{R} \mid D_{i+1,j+2}\right) P\left(\mathbf{y}_{j+1}^{j+2} \mid D_{i+1,j+2}, D_{i,j}\right) \\ &= P_d P\left(\mathbf{y}_{j+1}^{R} \mid D_{i+1,j}\right) + P_t P\left(\mathbf{y}_{j+2}^{R} \mid D_{i+1,j+1}\right) P\left(y_{j+1} \mid D_{i+1,j+1}, D_{i,j}\right) \\ &\quad + P_i P\left(\mathbf{y}_{j+3}^{R} \mid D_{i+1,j+2}\right) P\left(\mathbf{y}_{j+1}^{j+2} \mid D_{i+1,j+2}, D_{i,j}\right).\end{aligned}$$

Due to the definition of backward variable, the following equation is obtained,

$$\begin{aligned}\beta_i(j) = P_d\beta_{i+1}(j) + P_t\beta_{i+1}(j+1)P\left(y_{j+1} \mid, D_{i+1,j+1}, D_{i,j}\right) \\ + P_i\beta_{i+1}(j+2)P\left(\mathbf{y}_{j+1}^{j+2} \mid D_{i+1,j+2}, D_{i,j}\right).\end{aligned}$$

$P\left(\mathbf{y}_{j+1}^{j+2} \mid D_{i+1,j+2}, D_{i,j}\right) = 1/4$, and $P\left(y_{j+1} \mid, D_{i+1,j+1}, D_{i,j}\right) = G(x_i, y_j)$, then the ultimate recursion term is obtained,

$$\beta_i(j) = P_d\beta_{i+1}(j) + \frac{P_i}{4}\beta_{i+1}(j+2) + P_t\beta_{i+1}(j+1)\sum_{x_{i+1}} P(x_{j+1})G(x_{i+1}, y_{j+1}).$$

The initial conditions for calculating backward probabilities are,

$$\beta_T(j) = \begin{cases} 1, & \text{if } j = R, \\ 0, & \text{if } j \in \{0, 1, \ldots, R-1\}. \end{cases}$$

Then likelihoods terms, $P\left(\mathbf{y}_1^R \mid x_k\right)$, for $k \in \{1, \ldots, T\}$, and $x_k \in \{0, 1\}$ are calculated as follows:

$$\begin{aligned}P\left(\mathbf{y}_1^R \mid x_k\right) &= \sum_{n=0}^{\min(2k,R)} P\left(\mathbf{y}_1^R, D_{k,n} \mid x_k\right) \\ &= \sum_{n=0}^{\min(2k,R)} \Big(P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n} \mid x_k\right) + P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n-1} \mid x_k\right) \\ &\quad + P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n-2} \mid x_k\right) \Big).\end{aligned}$$

Let us examine the term $P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n-2} \mid x_k\right)$, and expand as follows:

$$\begin{aligned} P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n-2} \mid x_k\right) &= P\left(\mathbf{y}_1^{n-2}, D_{k-1,n-2} \mid x_k\right) \\ &\quad P\left(\mathbf{y}_{n-1}^{n}, D_{k,n} \mid \mathbf{y}_1^{n-2}, x_k, D_{k-1,n-2}\right) \\ &\quad P\left(\mathbf{y}_{n+1}^{R} \mid D_{k,n}, \mathbf{y}_1^{n}, x_k, D_{k-1,n-2}\right) \\ &= P\left(\mathbf{y}_1^{n-2}, D_{k-1,n-2}\right) P\left(\mathbf{y}_{n-1}^{n}, D_{k,n} \mid D_{k-1,n-2}, x_k\right) \\ &\quad P\left(\mathbf{y}_{n+1}^{R} \mid D_{k,n}\right) \\ &= \frac{P_i}{4}\alpha_{k-1}(n-2)\beta_k(n). \end{aligned}$$

Secondly, let us examine the term $P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n-1} \mid x_k\right)$, and expand as follows,

$$\begin{aligned} P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n-1} \mid x_k\right) &= P\left(\mathbf{y}_1^{n-1}, D_{k-1,n-1} \mid x_k\right) P\left(\mathbf{y}_n, D_{k,n} \mid \mathbf{y}_1^{n-1}, x_k, D_{k-1,n-1}\right) \\ &\quad P\left(\mathbf{y}_{n+1}^{R} \mid D_{k,n}, \mathbf{y}_1^{n}, x_k, D_{k-1,n-1}\right) \\ &= P\left(\mathbf{y}_1^{n-1}, D_{k-1,n-1}\right) P\left(y_n \mid D_{k,n}, D_{k-1,n-1}, x_k\right) \\ &\quad P\left(D_{k,n} \mid D_{k-1,n-1}\right) P\left(\mathbf{y}_{n+1}^{R} \mid D_{k,n}\right) \\ &= P_t G(x_k, y_n)\alpha_{k-1}(n-1)\beta_k(n). \end{aligned}$$

The term $P\left(\mathbf{y}_1^R, D_{k,n}, D_{k-1,n} \mid x_k\right)$ is similarly derived. Then, the likelihood term $P\left(\mathbf{y}_1^R \mid x_k\right)$ for $x_k \in \{0, 1\}$ is calculated as follows:

$$\begin{aligned} P\left(\mathbf{y}_1^R \mid x_k\right) = &\sum_{n=0}^{\min(2k,R)} P_d \alpha_{k-1}(n)\beta_{k-1}(n) \\ &+ \sum_{n=1}^{\min(2k,R)} P_t G(x_k, y_n)\alpha_{k-1}(n-1)\beta_k(n) \\ &+ \sum_{n=2}^{\min(2k,R)} \frac{P_i}{4}\alpha_{k-1}(n-2)\beta_k(n). \end{aligned}$$

Some numerical instabilities occur as the probabilistic implementation of the BCJR algorithm is used. Hence, a logarithmic implementation of the BCJR algorithm is preferable. Let $\tilde{\alpha}_k(n) = \log \alpha_k(n)$ and $\tilde{\beta}_k(n) = \log \beta_k(n)$, the logarithmic domain equivalents of $\alpha_k(n)$ and $\beta_k(n)$. Logarithm of the forward recursion is

taken, that is,

$$\log \alpha_i(j) = \log \left(P_d \alpha_{i-1}(j) + P_t \alpha_{i-1}(j-1) \sum_{x_i} P(x_i) G(x_i, y_j) + \frac{P_i}{4} \alpha_{i-1}(j-2) \right).$$

Since $\tilde{\alpha}_k(n) = \log \alpha_k(n)$ and $e^{\log x} = x$, we can write,

$$\tilde{\alpha}_i(j) = \log \left(e^{\log P_d + \tilde{\alpha}_{i-1}(j)} + e^{\log P_t + \tilde{\alpha}_{i-1}(j-1) + \log\left(\sum_{x_i} P(x_i) G(x_i, y_j)\right)} + e^{\log \frac{P_i}{4} + \tilde{\alpha}_{i-1}(j-2)} \right).$$

Then, we use the function $\max^*(x, y) = \log(e^x + e^y) = \max(x, y) + \log\left(1 + e^{-|x-y|}\right)$ defined in [33]. Indeed, $\max^*(x, y, z) = \log(e^x + e^y + e^z) = \max^*\left(\max^*(x, y), z\right)$. Using this, we can express the final form of the recursion as follows:

$$\tilde{\alpha}_i(j) = \max^* \Bigg(\log P_d + \tilde{\alpha}_{i-1}(j),\ \log P_t + \tilde{\alpha}_{i-1}(j-1) + \log \left(\sum_{x_i} P(x_i) G(x_i, y_j) \right), \log \frac{P_i}{4} + \tilde{\alpha}_{i-1}(j-2) \Bigg).$$

Similarly, log-domain version of backward recursion is found as,

$$\tilde{\beta}_i(j) = \max^* \Bigg(\log P_d + \tilde{\beta}_{i+1}(j),\ \log P_t + \tilde{\beta}_{i+1}(j+1) + \log \left(\sum_{x_i} P(x_i) G(x_i, y_j) \right), \log \frac{P_i}{4} + \tilde{\beta}_{i+1}(j+2) \Bigg).$$

For message estimates, we use log-equivalent versions of forward and backward probabilities $\tilde{\alpha}_i(j)$ and $\tilde{\beta}_i(j)$. Taking logarithm of both sides, and using $\tilde{\alpha}_i(j)$ and $\tilde{\beta}_i(j)$, the equation becomes,

$$\begin{aligned} \log P\left(\mathbf{y}_1^R \mid x_i\right) = \log \Bigg(& \sum_{j=0}^{\min(2i,R)} e^{\log P_d + \tilde{\alpha}_{i-1}(j) + \tilde{\beta}_i(j)} \\ & + \sum_{j=1}^{\min(2i,R)} e^{\log P_t + \log G(x_i, y_j) + \tilde{\alpha}_{i-1}(j-1) + \tilde{\beta}_i(j)} \\ & + \sum_{j=2}^{\min(2i,R)} e^{\log \frac{P_i}{4} + \tilde{\alpha}_{i-1}(j-2) + \tilde{\beta}_i(j)} \Bigg). \end{aligned}$$

Since, $\max^*(x_1, x_2, \ldots, x_n) = \max^*\left(\max^*(x_1, x_2, \ldots, x_{n-1}), x_n\right)$, the above equation can be computed recursively. Then we can calculate LLR as follows:

$$L_i = \log P\left(x_i = 0 \mid \mathbf{y}_1^R\right) - \log P\left(x_i = 1 \mid \mathbf{y}_1^R\right).$$

An important note is that the algorithm depends on the perfect knowledge of the channel parameters P_i, P_d, and P_s. When the channel parameters are not precisely known, the algorithm's performance becomes suboptimal.

2.4 Deep Learning for Channel Coding

Various deep learning architectures have been developed for coding over different channels. From its inception, the adoption of neural networks for decoding has been an intriguing topic. Several decoder architectures based on multi-layer perceptrons (MLPs) have been proposed for one-shot decoding of different types of channel codes in [37]. The input layer of the network receives channel values, and the output layer of the network produces message estimates. Decoders based on MLPs have demonstrated that structured codes, such as polar codes, are learnable by neural networks, and generalizations to unseen codewords are possible. For short codeword lengths, the bit error rate (BER) performance of deep learning-based decoders matches that of MAP decoding. Various experiments are conducted in order to train these architectures efficiently. However, for moderate codelengths (codelengths over 100 bits), decoding with basic neural network architectures such as MLPs does not appear feasible due to the complex one-shot decoding network since a number of input layers of the network equals codeword length and input to the network is received channel values. In order to resolve this issue with MLPs, modified MLPs are adopted to decode various families of linear block codes [38, 39]. As an example, in [38], MLPs mimicking belief propagation algorithm with additional weights and having extra layers have shown that with basic modifications to architecture, BCH codes in the length of hundreds could be decoded by MLP and its error rate performance is close to the belief propagation performance.

Recurrent Neural Networks (RNNs) offer excellent performance in handling time-series data. As a result, the RNNs offer a potential solution for decoding of convolutional codes since main decoders for RNNs, namely the Viterbi algorithm and BCJR decoders, employ sequential decoding being similar to RNN

architecture. RNN variants, namely gated recurrent unit (GRU) [40] and long short-term memory (LSTM) [41], have further enhanced the performance of RNN by alleviating the vanishing gradient problem and allowing the network to investigate further the relationship between the data separated by large time steps. Hence, networks based on LSTM and GRU architectures are proposed for decoding convolutional codes demonstrating a close to optimal error performance obtained by MAP decoders [42]. These networks do not only achieve optimal performance, but they also generalize to codeword lengths over large block lengths despite being only trained with small codeword lengths. GRUs replace the BCJR blocks in iterative decoding in the work [43]. Moreover, decoders based on the transformer architecture have been demonstrated successfully for different channel codes by combining the attention mechanism and the code structure [44]. These approaches offer decoding solutions for general linear block codes.

There are reinforcement learning (RL) based approaches to code constructions. The frozen bit locations in polar codes and polarization-adjusted convolutional (PAC) codes, are crucial factors for error rate performance in a specific decoder; hence, they must be properly determined. In [45, 46], the frozen bit locations of polar and PAC codes [47] for successive cancellation list decoders are determined by letting the agent in the RL environment explore possible combinations of frozen bit locations effectively. These papers use the basic temporal difference algorithms. In [48], authors adopt the similar RL methods to construction of parity check matrices of linear block codes. The authors in [49] employ Q-learning and its deep learning variant in order to determine iterative scheduling for bit-flipped decoding. Authors in [50] propose an iterative RL-based scheduling for the sum-product algorithm of LDPC codes.

2.5 Chapter Summary

We began the chapter by discussing the practical reasons for using channels with synchronization errors and their corresponding mathematical models. Next, we

explored the lower and upper bounds on the capacity of channels with i.i.d. synchronization errors, as presented in the literature. We reviewed the primary channel codes utilized for these channels and subsequently described concatenated codes, where the inner code is a known sequence shared with the receiver, and the outer code is a robust channel code. We explained our motivation for selecting marker codes in our context. The chapter also examined the prevalent methods in the literature for decoding these codes. Finally, we explored current deep-learning approaches for decoding channel codes, highlighting their potential as viable candidates.

Chapter 3

Bit Level Deep Learning Based Decoders for Concatenated Codes over Insertion and Deletion Channels

In this chapter, we develop deep learning-based estimators and decoders for concatenating an outer low-density parity-check (LDPC) or convolutional code with an inner marker code used over channels with deletions and substitution errors. Our aim is to provide alternative methods for decoding inner codes and to address the limitations of conventional methods, particularly, their dependence on knowing the channel parameters for optimal performance. Our contributions include the development of:

- a bi-directional gated recurrent unit (BI-GRU) estimator for the bit log-likelihood ratios (LLRs) for the inner marker code,
- one-shot decoder for serial concatenation of convolutional and marker codes via a complete BI-GRU based architecture.

While these solutions are developed for deletion and substitution channels, they can easily be extended to other channels with synchronization errors, including insertion and deletion channels by simply training the networks with the new models.

The chapter is organized as follows. In Section 3.1, the channel model, specifically the deletion/substitution channel, is introduced. In Section 3.2, the proposed system model is presented with the BI-GRU architecture as a deep learning-based estimator for bit LLRs. In Section 3.2.1, the bi-directional recurrent neural network (BI-RNN) deep learning architecture is discussed in detail, as it represents the general sequential deep learning framework compared to BI-GRU. Subsequently, the BI-GRU architecture is explained in detail in Section 3.2.2. Numerical results are provided in Section 3.3. The complexity of BI-GRU decoders is discussed in Section 3.4. Finally, the chapter is concluded in Section 3.5.

3.1 Deletion/Substitution Channel Model

We consider transmission over binary channels susceptible to deletions, and bit flips, modeling synchronization errors. Let P_d and P_s be the deletion and substitution probabilities of the channel, respectively. That is, each bit is either deleted with probability P_d, transmitted incorrectly with probability $(1-P_d)P_s$, or transmitted correctly with probability $(1-P_d)(1-P_s)$. The deletions and substitutions are independent of the different uses of the channel. Note that while we consider deletion/substitution channels, there is no fundamental restriction. We mainly focus deletion/substitution channels because they provide easier conditions for networks to train.

3.2 Proposed Decoding Scheme Employing BI-GRUs

We consider two different setups. In the first setup, we serially concatenate an LDPC (or a convolutional code) with a marker code, and a BI-GRU architecture is developed to estimate the LLRs of the bits input to the marker code. An inner marker code is adopted to regain the synchronization disrupted by the deletion channel. By using such a marker sequence, the receiver regains the synchronization. LDPC codes are adopted as the outer codes to improve the bit-error performance of the system.

Fig. 3.1 illustrates the block diagram of the first setup. In the second setup, we focus on the serial concatenation of a convolutional code with a marker code and develop BI-GRU architectures that perform one-shot decoding. That is, two BI-GRU architectures are combined to estimate the LLRs and decode the message bits in one step. Fig. 3.2 depicts the block diagram of the second setup, where the entire receiver is replaced by BI-GRU architectures.

We adopt a serially concatenated scheme as the specific channel coding approach in which the outer code is either an LDPC or a convolutional code, and the inner code is a marker code. Message vector $\mathbf{m} = (m_1, \ldots, m_k)$ of length k where $m_t \in \{0, 1\}$ is encoded via the outer code, and then the encoded sequence $\mathbf{c} = (c_1, \ldots, c_n)$ of length n where $c_t \in \{0, 1\}$ is interleaved in the bit-level, resulting in $\mathbf{c}^\pi$. Interleaving is employed in order to decorrelate message bits in order to meet the assumption that message bits are uncorrelated at the output of the channel. The rate of outer code, r_o, is k/n. Then, a pre-selected marker sequence with N_m bits is inserted after every N_c coded bits in $\mathbf{c}^\pi$, assuming n divides N_c, resulting in $\mathbf{x} = (x_1, \ldots, x_T)$ where T denotes the total number of transmitted bits where $T = n + N_m(n/N_c)$. If n, the codeword length of c, does not divide N_c, then one can simply pad 0's at the end of c until total number of bits of c divide N_c without loss of generality. The rate of marker code is, $r_m = N_m/(N_c + N_m)$. The rate of the overall code is $r = r_o r_m = \frac{N_c k}{(N_c+N_m)n}$. In a summary, marker codes are utilized to regain synchronization while LDPC codes account for the

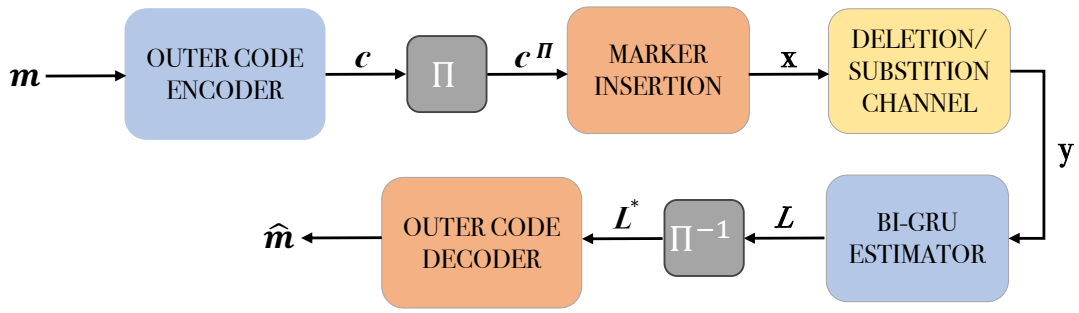

Figure 3.1: The block diagram for the first setup.

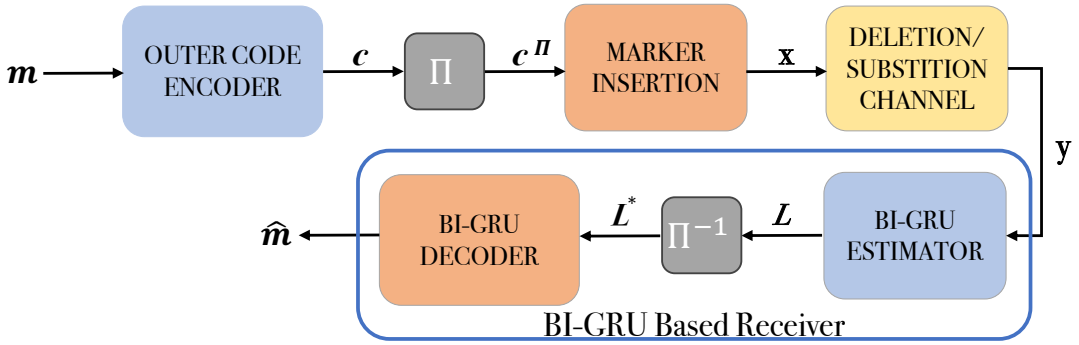

Figure 3.2: The block diagram for the second setup.

improvement of performance in terms of bit error rate and frame error rate.

The codeword is transmitted over a deletion/substitution channel. The number of received bits at receiver is R, being different than T due to deletions, insertions and the received sequence is $\mathbf{y}_1^R = (y_1, \ldots, y_R)$. The proposed BI-GRU-based architecture estimates the LLR of each transmitted bit, denoted by $\mathbf{L} = (L_1, \ldots, L_T)$. The estimated LLRs are first de-interleaved, and then those corresponding to marker bits are removed since they are irrelevant for the outer code decoder, resulting in the vector $\mathbf{L}^* = (L_1, \ldots, L_n)$. $\mathbf{L}^*$ is passed to the outer code decoder. A sum-product algorithm-based decoder can be used for LDPC codes, or the Viterbi algorithm can be employed for convolutional codes to estimate the message vector $\hat{\mathbf{m}} = (\hat{m}_1, \ldots \hat{m}_k)$. Other alternatives for the outer code decoder are also possible. For instance, we employ a BI-GRU architecture to estimate the message vector for the second setup [42].

3.2.1 Bi-directional RNN

In this section, we give details about the BI-RNN architecture and explain why we employ it for our problem rather than other deep learning architectures. We select the RNN architecture to estimate LLRs of the bits encoded by the outer codes because it can handle variable-length sequences produced at the output of a deletion channel. The basic deep learning architecture known as multi-layer perceptron (MLP) requires fixed-length inputs; therefore, they are not suitable for our problem. Moreover, the performance of MLPs is limited by the codeword lengths. This general phenomenon is also underscored in [37], which examines the limitations of MLPs for decoding channel codes.

Also note that, the BCJR algorithm explained in Section 2.3 resembles the structure of RNNs. This algorithm, also known as the forward and backward algorithm, uses bidirectional computation for its forward and backward variable calculations, which are used to estimate the LLRs. Hence, the structure of BI-RNN resembles the structure of the maximum a posteriori (MAP) detector realized by the forward-backward recursions. This phenomenon was examined in [42], which employs BI-RNNs for decoding of convolutional codes with the aim of replacing the BCJR algorithm.

Let us examine the architecture of BI-RNNs. A BI-RNN consists of two independent RNN architectures, forward and backward RNN, enabling the network to capture information effectively in both forward and backward directions. In order to understand the overall structure of a BI-RNN, we first give details of a forward RNN layer. In a basic forward RNN at layer n, where $n = 1, \ldots, N$, the hidden state at a given time t, $\overrightarrow{h}_t^{(n)}$, represents the prior information vector with size $d_{RNN} \times 1$ where d_{RNN} is a hyperparameter of the RNN architecture. Another input to a forward RNN at layer n and at timestep t is $\overrightarrow{h}_t^{(n-1)}$, the hidden state at the $(n-1)^{th}$ layer and timestep t. A special case is $\overrightarrow{h}_t^{(0)}$ which is the input to the overall system at time step t, $\overrightarrow{h}_t^{(0)} = x_t$. RNNs can be stacked, and the outputs of previous layers can be fed to the upper layer.

Let $\overrightarrow{W}_{rec}^{(n)}$ and $\overrightarrow{W}_{in}^{(n)}$ denote the recurrent weight matrix and the input weight

matrix of forward RNN at layer n, respectively. Sizes of these matrices are $d_{RNN} \times d_{RNN}$ for $n = 2, \ldots, n$ and $d_{RNN} \times d_x$ for $n = 1$ where d_x denotes the length of input to overall system. We can compute $\overrightarrow{h}_t^n$ as follows where f is a nonlinear activation function:

$$\overrightarrow{h}_t^{(n)} = f(\overrightarrow{W}_{rec}^{(n)} \overrightarrow{h}_{t-1}^{(n)} + \overrightarrow{W}_{in}^{(n)} \overrightarrow{h}_t^{(n-1)}). \tag{3.1}$$

In a backward RNN, the hidden state at a given time t and at layer n represents the information about the sequence to the right of the current input, $\overleftarrow{h}_t^{(n)}$. Let $\overleftarrow{W}_{rec}^{(n)}$ and $\overleftarrow{W}_{in}^{(n)}$ denote the recurrent weight matrix and the input weight matrix of backward RNN at layer n, respectively. The sizes of these matrices are the same as those of the matrices in the forward layers. We can compute $\overleftarrow{h}_t^{(n)}$ as follows:

$$\overleftarrow{h}_t^{(n)} = f(\overleftarrow{W}_{rec}^{(n)} \overleftarrow{h}_{t+1}^{(n)} + \overleftarrow{W}_{in}^{(n)} \overleftarrow{h}_t^{(n-1)}). \tag{3.2}$$

A layer of BI-RNN [51] combines a forward and a backward RNN described above. A linear layer with a sigmoid activation function is added on top of the final BI-RNN layer, which is N^{th} BI-RNN layer, to calculate the conditional probability of $x_t = 1$ given y where y denotes the channel output. Let W_o denote the output weight matrix of the network with a size $1 \times 2d_{RNN}$, σ be the sigmoid function and $[\,;\,]$ be the concatenation operator. We calculate the output of the network as $P(x_t = 1|y) = \sigma(W_o[\overrightarrow{h}_t^{(N)}; \overleftarrow{h}_t^{(N)}])$. We note that during the training process, $\overrightarrow{W}_{rec}^{(n)}$, $\overleftarrow{W}_{rec}^{(n)}$ for $n = 1, \ldots, N$ and W_o are learned. Then, the LLR of x_t, $L(x_t|y)$ for $t = 1, 2, \ldots, T$ is computed using:

$$L(x_t|y) \triangleq \log \frac{P(x_t = 0|y)}{P(x_t = 1|y)} = \log \frac{1 - \sigma(W_o[\overrightarrow{h}_t^{(N)}; \overleftarrow{h}_t^{(N)}])}{\sigma(W_o[\overrightarrow{h}_t^{(N)}; \overleftarrow{h}_t^{(N)}])}. \tag{3.3}$$

The problem of estimating LLRs in our case is nothing but a classification between whether the bit at timestep t is 0 or 1.

3.2.2 BI-GRU as an LLR Estimator

We implement the BI-GRU architecture as an LLR estimator for the marker code since they address the issues of vanishing and exploding gradients commonly

faced in RNNs, while also significantly enhancing memory capabilities compared to simple RNNs [40, 52]. GRUs are chosen instead of LSTMs since the latter is more challenging to train. The BI-GRU is an extension of BI-RNN and has two independent components that are similar to BI-RNN outlined in Section 3.2.1, namely forward and backward BI-GRU. Let $\odot$ denote the Hadamard Product where $x \odot y = (x_1, \ldots, x_n) \odot (y_1, \ldots, y_n) = (x_1 y_1, \ldots, x_n y_n)$; equations of a n^{th} forward GRU layer are as follows [40]:

$$
\begin{aligned}
\overrightarrow{z}_t^{(n)} &= \sigma\left(\overrightarrow{W}_z^{(n)} \cdot \overrightarrow{h}_t^{(n-1)} + \overrightarrow{U}_z^{(n)} \cdot \overrightarrow{h}_{t-1}^{(n)} + \overrightarrow{b}_z^{(n)}\right) \\
\overrightarrow{r}_t^{(n)} &= \sigma\left(\overrightarrow{W}_r^{(n)} \cdot \overrightarrow{h}_t^{(n-1)} + \overrightarrow{U}_r^{(n)} \cdot \overrightarrow{h}_{t-1}^{(n)} + \overrightarrow{b}_r^{(n)}\right) \\
\overrightarrow{\tilde{h}}_t^{(n)} &= \tanh\left(\overrightarrow{W}_h^{(n)} \cdot \overrightarrow{h}_t^{(n-1)} + \overrightarrow{U}_h^{(n)} \cdot (\overrightarrow{r}_t^{(n)} \odot \overrightarrow{h}_{t-1}^{(n)}) + \overrightarrow{b}_h^{(n)}\right) \\
\overrightarrow{h}_t^{(n)} &= \overrightarrow{z}_t^{(n)} \odot \overrightarrow{h}_{t-1}^{(n)} + (1 - \overrightarrow{z}_t^{(n)}) \odot \overrightarrow{\tilde{h}}_t^{(n)}
\end{aligned}
$$

where $\overrightarrow{z}_t^{(n)}$ is the vector that controls how much new information is added to the hidden state, $\overrightarrow{\tilde{h}}_t^{(n)}$ is the vector that represents the new candidate information, and $\overrightarrow{r}_t^{(n)}$ is the vector that controls how much the previous state information is passed to the temporary variable $\overrightarrow{\tilde{h}}^{(n)}$.

Similarly, backward GRU equations are easily obtained as in (3.2).

$$
\begin{aligned}
\overleftarrow{z}_t^{(n)} &= \sigma\left(\overleftarrow{W}_z^{(n)} \cdot \overleftarrow{h}_t^{(n-1)} + \overleftarrow{U}_z^{(n)} \cdot \overleftarrow{h}_{t+1}^{(n)} + \overleftarrow{b}_z^{(n)}\right) \\
\overleftarrow{r}_t^{(n)} &= \sigma\left(\overleftarrow{W}_r^{(n)} \cdot \overleftarrow{h}_t^{(n-1)} + \overleftarrow{U}_r^{(n)} \cdot \overleftarrow{h}_{t+1}^{(n)} + \overleftarrow{b}_r^{(n)}\right) \\
\overleftarrow{\tilde{h}}_t^{(n)} &= \tanh\left(\overleftarrow{W}_h^{(n)} \cdot \overleftarrow{h}_t^{(n-1)} + \overleftarrow{U}_h^{(n)} \cdot (\overleftarrow{r}_t^{(n)} \odot \overleftarrow{h}_{t+1}^{(n)} + \overleftarrow{b}_h^{(n)}\right) \\
\overleftarrow{h}_t^{(n)} &= \overleftarrow{z}_t^{(n)} \odot \overleftarrow{h}_{t+1}^{(n)} + (1 - \overleftarrow{z}_t^{(n)}) \odot \overleftarrow{\tilde{h}}_t^{(n)}
\end{aligned}
$$

An MLP with N_{MLP} layers with dimensions d_{MLP} is added at the top of the final layer of BI-GRU, and then (3.3) can be used to estimate the bit LLRs as in the case of BI-RNNs. Fig. 3.3 shows the block diagram of BI-GRU architecture that estimates bit LLRs.

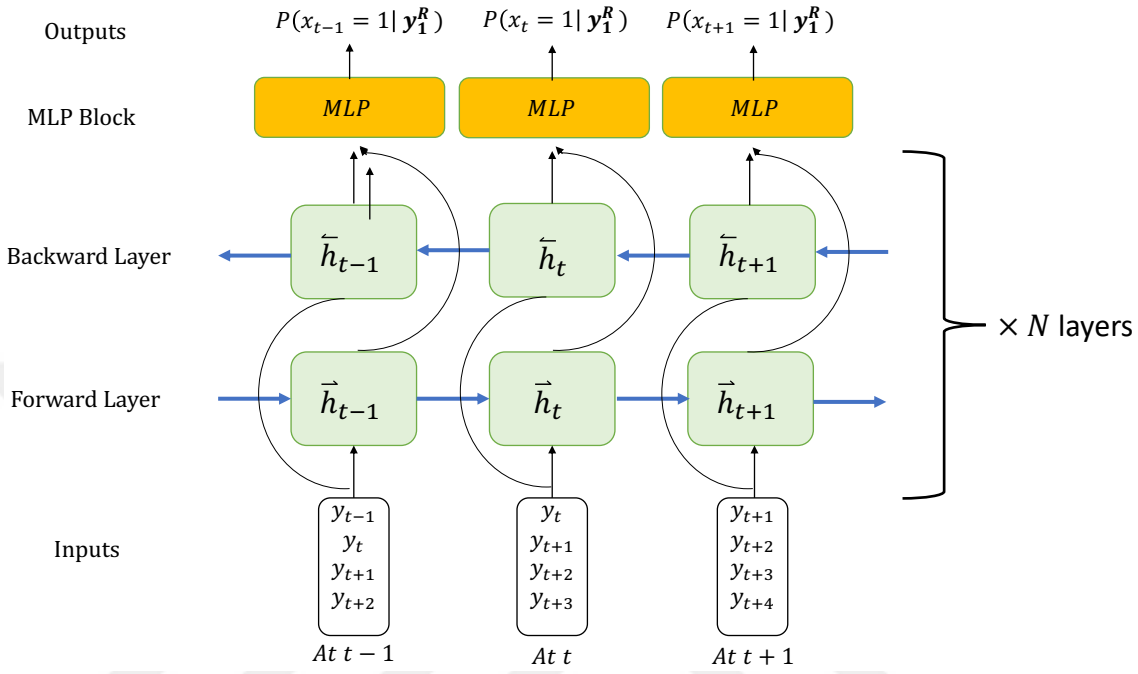

Figure 3.3: The block diagram for BI-GRU estimator with first training approach.

3.2.3 One Shot Decoding of Convolutional Codes Concatenated with Markers

As an alternative approach, we utilize the BI-GRU architecture as the decoder for the outer code in the second setup, employed in [42]. At time step t, two consecutive LLRs (L^*_{2t-1}, L^*_{2t}) of the vector $\mathbf{L}^*$ are input to the BI-GRU network and the output is the message estimate $\hat{m}_t$ for $t = 1, \ldots, k$. A linear layer with a sigmoid activation function is added on top of the final BI-GRU layer to estimate message bits. The decoder is illustrated in Fig. 3.4.

3.2.4 Implementation Details

We now describe the training procedure of the BI-GRU architecture for the proposed setups. The BI-GRU networks are trained with a supervised learning approach: For the BI-GRU based estimator, labels are the original transmitted sequences $\mathbf{x}$ over a deletion/substitution channel, and the inputs are the received

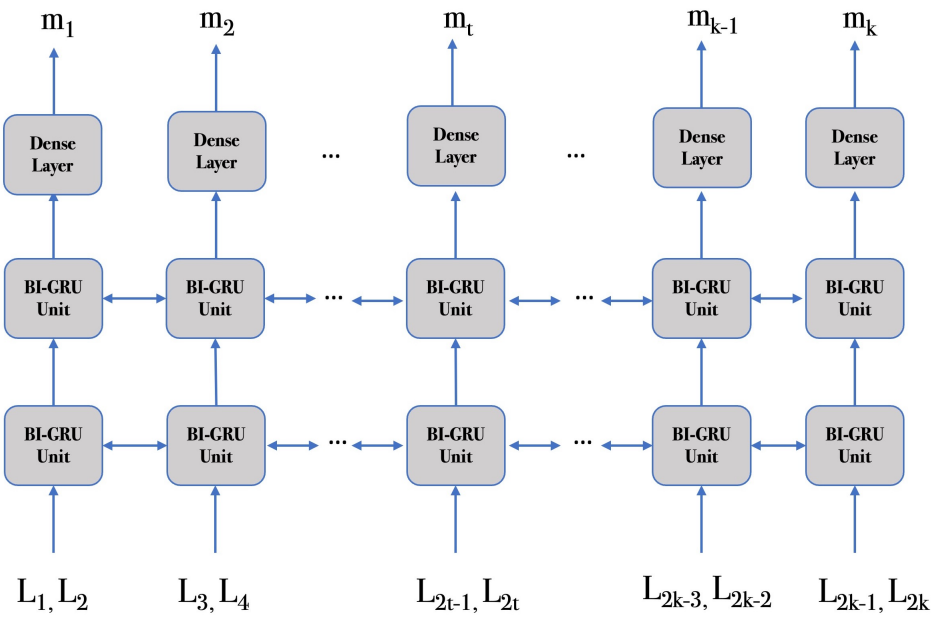

Figure 3.4: Block diagram of BI-GRU decoder.

sequence $\mathbf{y}$ transformed by $-2\mathbf{y}+1$ whereas for the BI-GRU decoder, labels are message sequences $\mathbf{m}$ and inputs are the LLR vectors $\mathbf{L}^*$ obtained from the BI-GRU based estimator. We do not use a pre-determined data set, instead, we generate each mini-batch randomly during training. In other words, we generate random message bits $\mathbf{m}$ (labels), encode them using an outer encoder to produce $\mathbf{x}$, which are then transmitted through a deletion/substitution channel. This process yields either $\mathbf{y}$ (inputs to the estimator) or $\mathbf{L}^*$ (inputs to the decoder), obtained by estimating LLRs from $\mathbf{y}$. Note that, the BI-GRU based decoders and estimators are trained independently.

We use the Adam optimizer [53] and a staircase learning decay [54], for which in each S step of the Adam optimizer, the learning rate decays exponentially with parameter D during training. If not stated, we use the default hyperparameters of the Adam optimizer. Batch normalization layers are added between the BI-GRU layers to train the network smoothly for both the estimator and the decoder [55].

3.3 Numerical Results

As a first example, we use a regular, rate 1/2 LDPC code with $n = 204$ and $k = 102$ (taken from [56]). The maximum iteration number of the sum-product decoder is set to 100. We use a two-bit marker $[0, 1]$ and choose N_c as 5 and 10, which results in code rates of 0.358 and 0.4167, respectively. We train two networks for $30,000$ steps, fixing $P_d = 0.05$ and $P_s = 0$. The initial learning rate is chosen as $9 \cdot 10^{-5}$ for the Adam optimizer, and we use a mini-batch size of 16. The overall input length is T, and since $R \leq T$, a total of $T - R$ zeros are padded for training and inference. From our numerical experiments, two consecutive bits after t including the current one, (y_t, y_{t+1}) are input to the BI-GRU at time step t since it performs better than giving only y_t as an input. We set $D = 0.95$ and, $S = 1000$ as detailed in Section 3.2.4. We use gradient clipping with 0.1 to avoid the vanishing gradient problem. LLRs obtained from the BI-GRU estimator are clipped in between $[-10, 10]$ during testing. The loss function is chosen as mean-squared error despite it is not common for classification taks. Parameters of the trained BI-GRUs are given in the first and second columns in Table 3.1.

Fig. 3.5 and Fig. 3.6 show the error rate performance of the proposed decoders for the first setup. We observe that the proposed decoders generalize well over the deletion probabilities in the range of $0.02 - 0.07$ despite being trained only with $P_d = 0.05$. The performance gap between the baseline and proposed decoder is smaller for $N_c = 10$. For instance, when $N_c = 5$, a bit error rate (BER) of 10^{-3} is achieved at a deletion probability of 0.04 for the proposed decoder and 0.05 for the baseline decoder. When $N_c = 10$, the same BER is obtained at a deletion probability of 0.025 for the proposed decoder and at around 0.027 for the baseline.

As a second example, we trained a network to be robust to different deletion and substitution probabilities. We use the same LDPC code and the marker sequence as in the first example and choose $N_c = 10$. The network is trained by mixing deletion probabilities in $[0.01, 0.1]$ with 0.01 increments and mixing substitution probabilities in $[0, 0.1]$ with 0.01 increments. In this case, unlike the

first approach, the bits before timestep t, including the current one, $(y_1, \ldots, y_t)$, are input to the BI-GRU at time step t. As stated earlier, there is no fixed method for giving inputs and each has its own advantage. Leter on, we compare these input methods and shows its differences. The initial learning rate is chosen as $9 \cdot 10^{-4}$, and we use a mini-batch size of 16. The loss function is chosen as binary cross entropy (BCE). In this case, since the input size is increased, we use smaller models. Parameters of the robust BI-GRU are given in the third column in Table 3.1.

	Estimator 1	**Estimator 2**	**Decoder**
N_c	10	10	-
$N_{\text{BI-GRU}}$	6	4	2
$d_{\text{BI-GRU}}$	1024	128	400
N_{MLP}	3	3	2
d_{MLP}	[128,32,1]	[128,32,1]	[32,1]
Learning Rate	9×10^{-4}	9×10^{-4}	3×10^{-4}
Batch Size	16	16	32
Steps	30000	30000	5000

Table 3.1: Parameters of trained BI-GRUs as an estimator or a decoder.

We also note that choosing proper training parameters of P_d is highly important and affects inference performance very well. There is no guideline for choosing the right values, however we advise you to adhere with the channel conditions you try while inference. For example, if you want to

An important point is that the baseline approach employing the BCJR algorithm uses the deletion and substitution probabilities of the channel to calculate the LLR estimates, while the BI-GRU based estimator does not. Fig. 3.7 compares the performance of the proposed decoder with the baseline over a range of substitution probabilities for a deletion/substitution channel with $P_d = 0.03$, and Fig. 3.8 compares the performance of the proposed decoder with the baseline over a range of deletion probabilities when $P_s = 0.05$. For both figures, we assume that the channel parameters are not known accurately at the receiver. The baseline decoder estimates the deletion probability using $\hat{P}_d = \frac{T-R}{T}$. Four different results are produced by taking the substitution probabilities used at the receiver as {0,

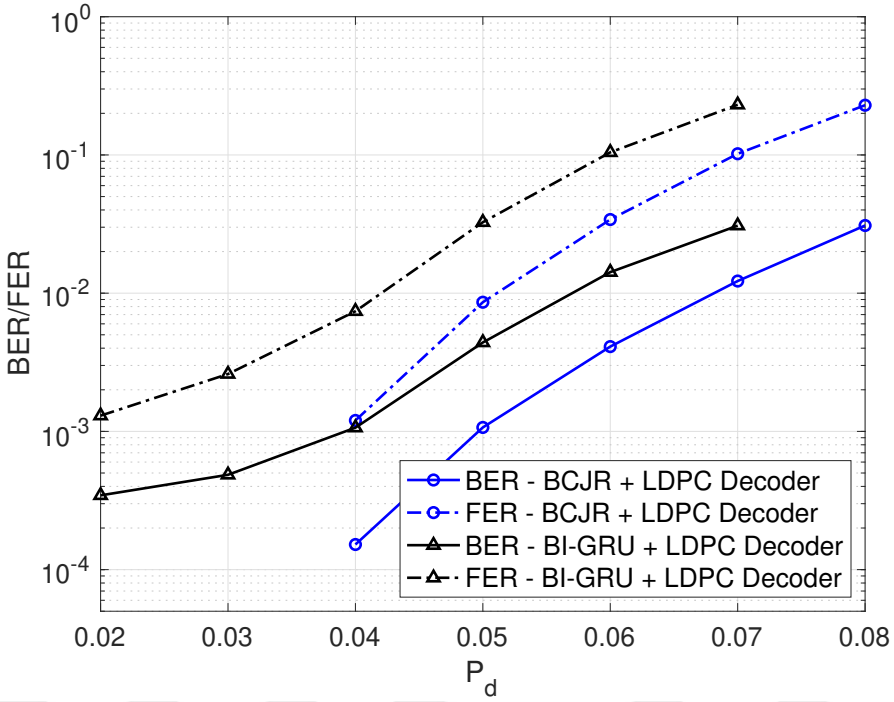

Figure 3.5: BER/FER as a function of the deletion probability for the first setup with BI-GRU as an estimator when $N_c = 5$.

0.03, 0.05, 0.1}. We observe that the proposed BI-GRU based decoder exhibits improved performance in some cases. Specifically, when the baseline decoder uses a substitution probability of 0, for a range of deletion probabilities, the proposed decoder outperforms the BCJR algorithm results. Notably, in terms of the frame error rate (FER) performance, the proposed decoders outperform the mismatched decoder for a wider range of channel parameters.

We also provide an example where we replace the decoder of the outer convolutional code with the fully BI-GRU-based decoder, detailed in Section 3.2.3. We use a rate 1/2, $(5, 7)_{octal}$ convolutional code with $n = 210$, and $k = 105$. The parameters of the BI-GRU decoder for the convolutional code are given in the fourth column of Table 3.1. We train the BI-GRU decoder for 5000 steps with an initial learning rate of $3 \cdot 10^{-4}$. We do not apply any learning rate decay for training in this case. We use the same BI-GRU estimator previously used for the first setup, whose parameters are given in the second column of Table 3.1. We

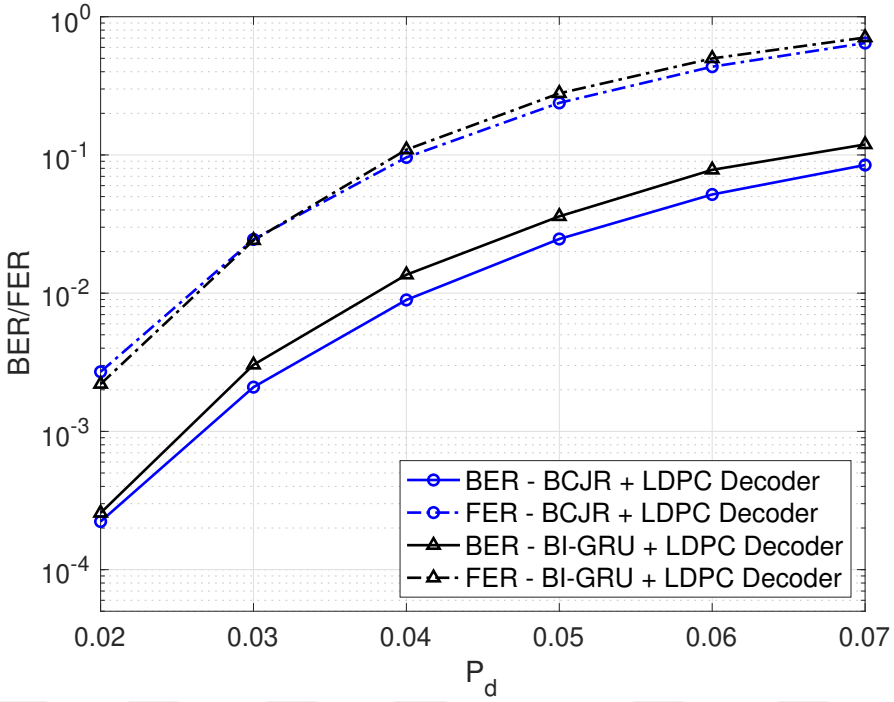

Figure 3.6: BER/FER as a function of the deletion probability for the first setup with BI-GRU as an estimator when $N_c = 10$.

consider two outer decoders as baselines: Viterbi algorithm with soft-decision decoding (SDD) and with hard-decision decoding (HDD). For the SDD case, LLRs of the BCJR algorithm-based estimator are used as if they are the outputs of an additive white Gaussian noise (AWGN) channel, and a correlation metric is used. For the case of the Viterbi algorithm with HDD, the LLRs of the estimator (it can be BI-GRU based or BCJR-algorithm based) are mapped to hard decision values: 0 if $L(x_t = 0|y) \geq 0$ and 1 if $L(x_t = 0|y) < 0$, which are then input to the Viterbi algorithm.

Fig. 3.9 depicts the resulting error rates. We observe that the fully BI-GRU-based decoder performs similarly to the baseline decoder when the Viterbi algorithm with SDD is used as the outer decoder and a BCJR algorithm is used as the estimator. Given that the training objective of the BI-GRU estimator is binary classification, there may be enhanced compatibility with the Viterbi algorithm using HDD; the decoder using the BI-GRU estimator along with the Viterbi algorithm employing HDD performs better than the baseline approach paired with

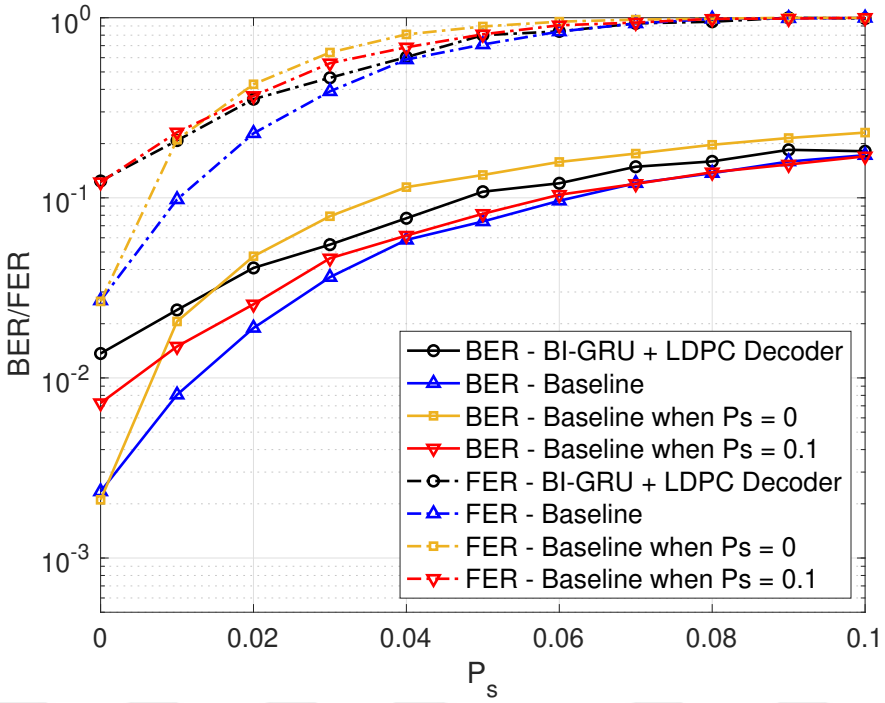

Figure 3.7: BER/FER as a function of substitution probability for a channel with $P_d = 0.03$.

the same outer decoding algorithm for all the deletion probabilities considered.

Finally, we want to comment on the numerical results of the recent work [32]. The results in [32] are compatible with ours: the BI-GRU-based estimators perform similarly to our trained decoders regarding their performance difference with the baseline decoders realized using forward-backward equations. The model-based decoders in [32] utilizing a forward-backward algorithm aimed to estimate P_d, P_s, and P_i, perform better than the BCJR-algorithm based decoders when the channel parameters are not known to the receiver. The authors also experiment with the burst insertion/deletion channels of which the deletions and insertions are not independent for different uses and show that without knowing the channel model, one may develop BI-GRU-based decoders and argue that their performance in terms of error rates is superior to those of the BCJR-based decoders.

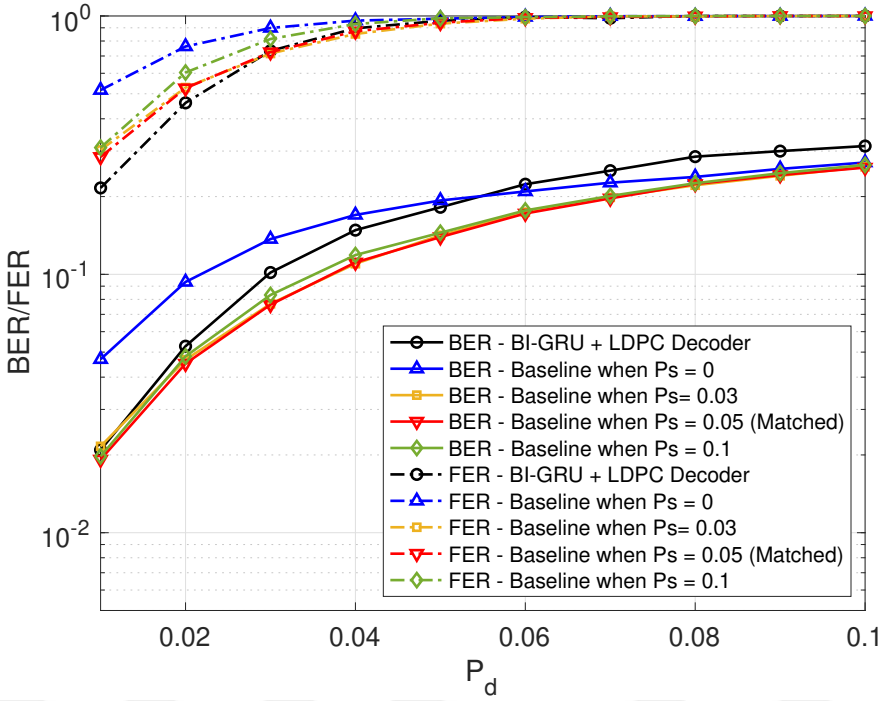

Figure 3.8: BER/FER as a function of deletion probability for a channel with $P_s = 0.05$.

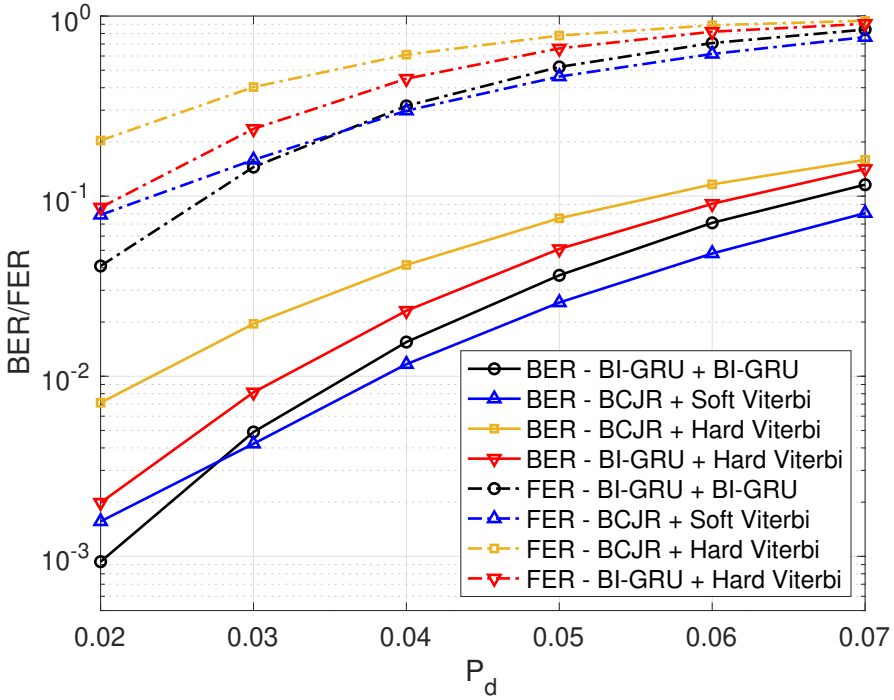

Figure 3.9: BER and FER as functions of deletion probability, with no substitution errors, with a convolutional as outer code.

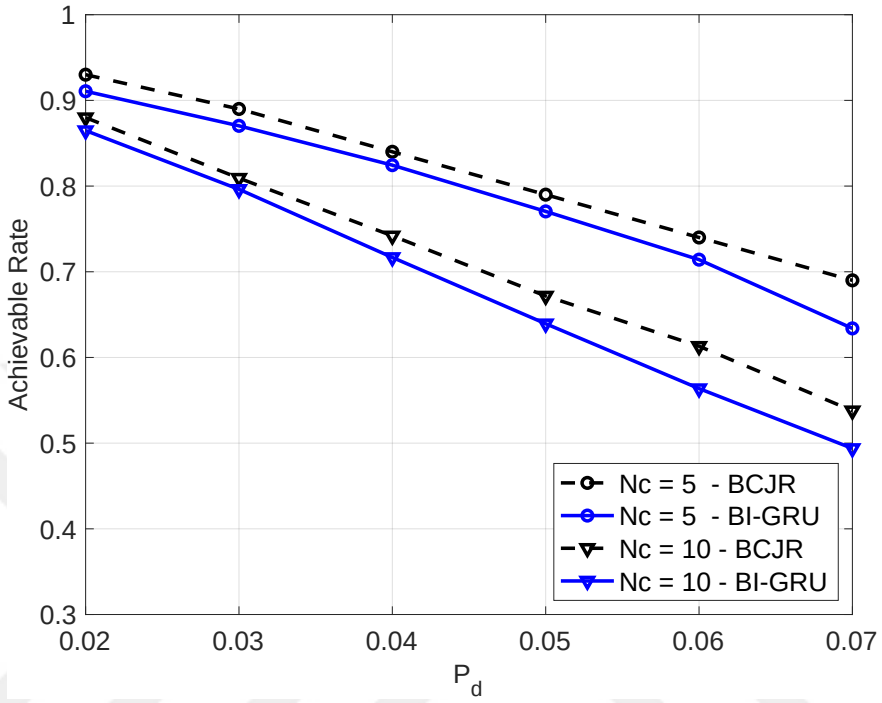

Figure 3.10: Achievable rates for BI-GRU network based on LLR estimates for marker codes.

3.3.1 Achievable Rate Estimation

We now provide some achievable rate estimates based on the LLR estimates of the BI-GRU network for marker codes. Since the mutual information cannot be computed in closed form due to the presence of the interleaver, it is evaluated through Monte Carlo simulations [31]. Therefore, histograms of the distributions $p(y), p(y|x = 0)$ and $p(y|x = 1)$ are obtained with a large number of channel realizations. By using the histograms, the achievable rate is calculated by plugging the LLR estimates into mutual information expression, $I(X;Y) = H(Y) - H(Y|X)$. Fig. 3.10 shows the results for BI-GRU based achievable rate estimation results compared to the baseline estimator computed via a BCJR algorithm.

For the baseline decoder, the histogram bin length is set to 0.25 with limits of $[-30, +30]$, while for the BI-GRU-based estimator, the bin length is 1 with the same limits. The achievable rates of the proposed BI-GRU estimator are

compared with those of the baseline decoder for $N_c = 5$ and $N_c = 10$ in Fig. 3.10. Our results show that the BI-GRU-based estimator offers achievable information rates similar to those of the MAP detector.

3.4 Complexity Analysis

For a BI-GRU architecture with d_{GRU} units, we have approximately $20d_{GRU}^2$ multiplications and additions. For all time instances, with a total of T time steps and N layers, there are approximately $20TNd_{GRU}^2$ multiplications and additions.

At each time instance k, approximately $8R$ multiplications and additions are needed for baseline approach. Since there are T time instances, where $k = 1, \ldots, T$, the total number of operations is approximately $8RT$. Since T and R are both close $R \approx T$, the total number of additions and multiplications is approximately $8T^2$.

The hyperparameter d_{GRU} in BI-GRU can be chosen smaller than T when T is large and closer to n for smaller codeword sizes, resulting in different complexity profiles. Therefore, when d_{GRU} is chosen close to T, the complexities of both methods become comparable. For larger codeword lengths, d_{GRU} can be chosen smaller than T, leading to a reduced complexity for BI-GRU.

The complexity of the BCJR algorithm can also be reduced. Several low-complexity implementations in the literature significantly lower the computational load, even for large codeword lengths. Similarly, the complexity of BI-GRU can be reduced using pruning techniques, which effectively reduce the computational cost, although such techniques were not considered in the above analysis.

3.5 Chapter Summary

In this chapter, we considered a concatenated coding setup where the inner code is a marker code, and the outer code is either an LDPC or a convolutional code. We proposed deep learning-based detectors for calculating the LLRs of the inner code and a joint deep learning-based decoder for calculating and decoding message bits when convolutional codes are used as outer codes. We utilized BI-GRUs due to their ability to handle variable-length outputs, which makes them suitable for insertion and deletion channels, and their resemblance to the BCJR algorithm. We explained the proposed estimator and our setup in detail. Our numerical results demonstrated that when channel conditions are not fully known to the receiver, our proposed decoder can outperform baseline methods. Achievable rate estimates with the proposed channel detectors were also obtained, and it was argued that the achievable rates are almost same as those with the baseline BCJR detector for the inner marker code. Complexity results for deep learning-based decoders were discussed, and it was argued that the complexity of BI-GRU-based decoders is manageable if the hyperparameters of the BI-GRU architecture are chosen to be as small as possible, and it can be further reduced by pruning the networks.

Chapter 4

Symbol-Level Deep Learning Based Decoders for Concatenated Codes over Insertion and Deletion Channels

This chapter proposes a novel deep learning-based method for decoding concatenated codes over insertion and deletion channels, focusing on symbol-level decoding. In Chapter 3, deep learning decoders were inspired by bit-level decoding methods, grounded in the forward-backward algorithm realized only by considering individual bits. However, in this chapter, the deep learning-based decoders are inspired by the novel symbol-level estimators introduced in [31]. Symbol-level decoders take into account the correlation among the bits at the receiver output, which are ignored by the bit-level decoders. When a bit is affected by the insertion and deletion channel, its adjacent bits are more influenced by this compared to further away bits. Therefore, symbol-level decoding provides enhancements over bit-level decoding by accounting for this correlation. In this method, symbols replace bits, and the posterior probabilities of these symbols are estimated using the maximum a posteriori (MAP) algorithm. While it is theoretically possible to create symbols of any length, prior work primarily focuses on 2-

and 3-bit symbol-level estimators, as the MAP detector becomes too complicated for implementation beyond these values.

This chapter's objectives are threefold: to replace the symbol-level MAP detector with a symbol-level deep learning-based detector to estimate the symbol probabilities, to improve the performance of bit-level deep learning-based detectors that failed to surpass the conventional methods using bit-level detectors, and to extend the limitations of previous symbol-level decoders to handle symbols longer than three bits.

In Section 4.1, the channel model, specifically the insertion and deletion channel, is revisited. In Section 4.2, the symbol-level MAP estimator is presented as an extension to the bit-level MAP solution and its details are given. In Section 4.3, the symbol-level based bi-directional gated recurrent unit (BI-GRU) estimators are proposed. Numerical results are provided in Section 4.4. The complexity of symbol-level BI-GRU decoders is discussed in Section 4.5. Finally, the chapter is concluded in Section 4.6.

4.1 Channel Model

We consider transmission over binary channels susceptible to deletions, insertions, and bit substitutions. Let P_d, P_i, and P_s represent the probabilities of deletion, insertion, and substitution events of the channel, respectively. Specifically, each bit in the sequence is either deleted with probability P_d, has two random bits inserted into the sequence at the time step of an insertion event with probability P_i, is transmitted incorrectly (substituted) with probability $(1 - P_d - P_i)P_s$, or is transmitted correctly with probability $(1 - P_d - P_i)(1 - P_s)$. Deletion, insertion, and substitution events are assumed to occur independently across different channel uses. This channel model is also known as the Gallager model used in [11], considering the insertion event as the insertion of two independent and identically distributed (i.i.d.) bits to sequence. The block diagram of this channel model is shown in Figure 4.1, which can also be treated as cascade of an insertion and

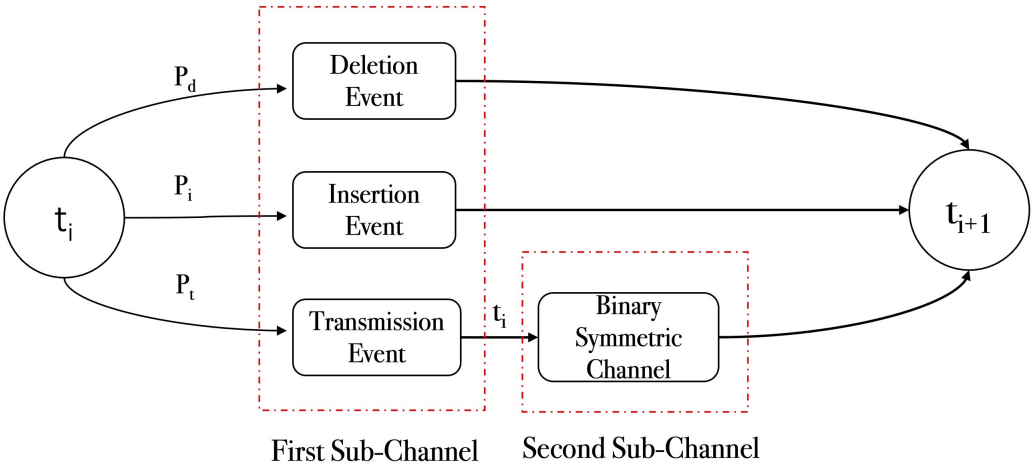

Figure 4.1: Gallager insertion/deletion channel model.

deletion channel followed by a binary symmetric channel (BSC).

4.2 Symbol-Level MAP Detector for Concatenated Codes for Insertion and Deletion Channels

Symbol-level MAP detectors for concatenated codes for insertion and deletion channels are proposed to account for the correlations among bits received over insertion and deletion channels. These correlations are ignored in bit-level estimation methods, e.g., as in [10]. The receiver's estimation accuracy can be improved by grouping bits into symbols and treating them jointly via a suitable MAP detector. The estimation accuracy improves as the symbol length increases, resulting in better error rate performance for the overall decoder.

We start this section by reviewing the overall system model proposed in [31]. A serially concatenated scheme is adopted in which the outer code is an low-density parity-check (LDPC) code and the inner code is a marker code. Message vector $\mathbf{m}$ is encoded through the outer (bit level) code, resulting in $\mathbf{c}$. The encoded sequence

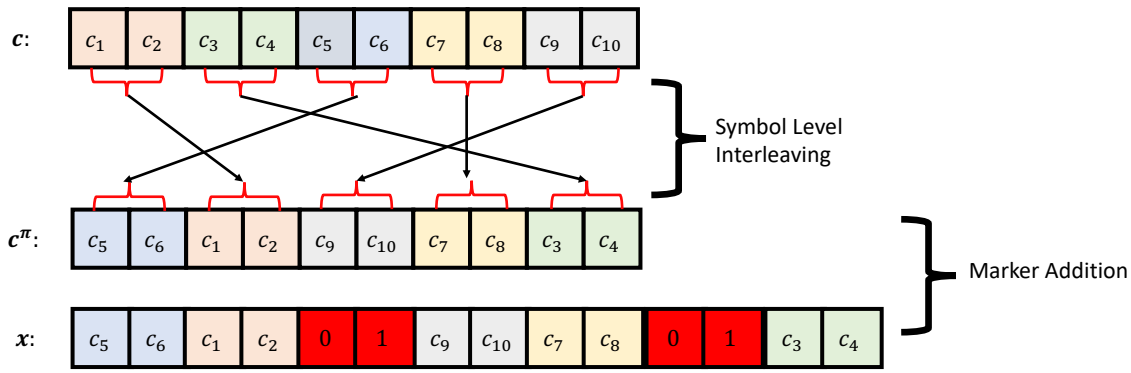

Figure 4.2: This case is for when $S = 2$, $N_c = 4$, $N_m = 2$, and the two-bit marker of $[0, 1]$ underscoring the two-bit interleaver.

$\mathbf{c}$ is grouped into symbols of length S and is expressed as $\mathbf{c} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{n/S})$, where each symbol is a group of S consecutive bits. For example, if $S = 2$, $\mathbf{s}_t = (c_{2t-1}, c_{2t})$, $t \in \{1, 2, \ldots, n/2\}$. The sequence $\mathbf{c}$ is interleaved in the symbol-level, resulting in $\mathbf{c}^\pi$. Then, the pre-selected marker with length N_m is inserted regularly into $\mathbf{c}^\pi$, resulting in the overall codeword, $\mathbf{x}$. The process is illustrated in Figure 4.2, which emphasizes symbol-level interleaving.

The codeword $\mathbf{x}$ is transmitted over the insertion and deletion channel and the received sequence is $\mathbf{y}_1^R$. Symbol-level MAP detector estimates the a-posteriori probabilities of the symbols encoded by the inner code, namely $P\left(\mathbf{s}_k \mid \mathbf{y}_1^R\right)$ for $k \in \{1, \ldots, T/S\}$. The posterior probabilities corresponding to marker indexes are discarded and the resulting sequence is de-interleaved. The resulting sequence is then input to a combination of the outer code and demapper module. Then, there is a local information exchange between the demapper and the bit-level outer code decoder. After some predetermined iterations, the outer code decoder yields hard decisions about the message bits.

Let us investigate the details of the symbol-level MAP detector. The binary event $D_{i,j}$ denotes that out of the first i symbols transmitted, and j bits are received. In other words, it denotes out of the first iS bits transmitted, j bits are received. The definitions of forward and backward probabilities aforementioned in Section 2.3 are used here as well. That is, the definitions of forward and

backward state variables are given as,

$$\alpha_i(j) \triangleq P\left(\mathbf{y}_1^j, D_{i,j}\right),$$
$$\beta_i(j) \triangleq P\left(\mathbf{y}_{j+1}^R \mid D_{i,j}\right),$$

where $i \in \{0, 1, 2, \ldots, T/S\}$ and $j \in \{0, 1, 2, \ldots, R\}$. A deeper understanding of these equations can be found in [31] and [57]. For completeness, we provide details of the forward-backward equations for the $S = 2$ case, along with a thorough explanation to introduce the fundamentals of symbol-level estimation in the following. Let us start with the definition of the forward equation, we decompose it using the total probability theorem,

$$\begin{aligned}
\alpha_i(j) &\triangleq P\left(\mathbf{y}_1^j, D_{i,j}\right) \\
&= \sum_{k=0}^{4} P\left(\mathbf{y}_1^j, D_{i,j}, D_{i-1,j-k}\right) \\
&= \sum_{k=1}^{4} P\left(\mathbf{y}_1^{j-k}, \mathbf{y}_{j-k+1}^j, D_{i,j}, D_{i-1,j-k}\right) + P\left(\mathbf{y}_1^j, D_{i,j}, D_{i-1,j}\right) \\
&= \sum_{k=1}^{4} P\left(\mathbf{y}_1^{j-k}, D_{i-1,j-k}\right) P\left(D_{i,j}, \mathbf{y}_{j-k+1}^j \mid \mathbf{y}_1^{j-k}, D_{i-1,j-k}\right) \\
&\quad + P\left(\mathbf{y}_1^j, D_{i-1,j}\right) P\left(D_{i,j} \mid \mathbf{y}_1^j, D_{i-1,j}\right),
\end{aligned}$$

where $\mathbf{y}_i^j = (y_i, y_{i+1}, \ldots, y_j)$. Recursion terms is obtained by realizing $P(\mathbf{y}_1^{j-k}, D_{i-1,j-k}) \triangleq \alpha_{i-1}(j-k)$,

$$\begin{aligned}
\alpha_i(j) &= \sum_{k=1}^{4} \alpha_{i-1}(j-k) P\left(D_{i,j}, \mathbf{y}_{j-k+1}^j \mid \mathbf{y}_1^{j-k}, D_{i-1,j-k}\right) \\
&\quad + \alpha_{i-1}(j) P\left(D_{i,j} \mid \mathbf{y}_1^j, D_{i-1,j}\right) \\
&= \sum_{k=1}^{4} \alpha_{i-1}(j-k) P\left(D_{i,j} \mid D_{i-1,j-k}\right) P\left(\mathbf{y}_{j-k+1}^j \mid D_{i,j}, D_{i-1,j-k}\right) \\
&\quad + \alpha_{i-1}(j) P\left(D_{i,j} \mid D_{i-1,j}\right).
\end{aligned}$$

$P\left(D_{i,j} \mid D_{i-1,j-4}\right) = P_i^2$ the transition happens only if two insertion events occur consecutively. Similarly, the transition between $D_{i-1,j-2}$ to $D_{i,j}$ happens either if either insertion and deletion event or two transmission events happen consecutively. The other terms follow with similar arguments. Using that intuition, we

label these transition events. For instance, if the first event is an insertion and the second is a transmission, we denote this transition as IT. We can continue,

$$\begin{aligned}\alpha_i(j) = {} & \alpha_{i-1}(j-4)P_i^2 P\left(\mathbf{y}_{j-3}^{j} \mid II\right) + \alpha_{i-1}(j-3)P(IT)P\left(\mathbf{y}_{j-2}^{j} \mid IT\right) \\ & + \alpha_{i-1}(j-3)P(TI)P\left(\mathbf{y}_{j-2}^{j} \mid TI\right) + \alpha_{i-1}(j-2)P\left(ID\right)P\left(\mathbf{y}_{j-1}^{j} \mid ID\right) \\ & + \alpha_{i-1}(j-2)P\left(DI\right)P\left(\mathbf{y}_{j-1}^{j} \mid DI\right) + \alpha_{i-1}(j-2)P\left(TT\right)P\left(\mathbf{y}_{j-1}^{j} \mid TT\right) \\ & + \alpha_{i-1}(j-1)P\left(TD\right)P\left(\mathbf{y}_{j} \mid TD\right) + \alpha_{i-1}(j-1)P\left(DT\right)P\left(\mathbf{y}_{j} \mid DT\right) \\ & + \alpha_{i-1}(j)P_d^2.\end{aligned}$$

The probability term, $P\left(\mathbf{y}_{j-3}^{j} \mid II\right) = P\left(\mathbf{y}_{j-3}^{j}\right) = 1/16$ since two insertion events occur and four bits are randomly inserted into the sequence. Similarly, $P\left(\mathbf{y}_{j-1}^{j} \mid DI\right) = P\left(\mathbf{y}_{j-1}^{j} \mid ID\right) = P\left(\mathbf{y}_{j-2}^{j}\right) = 1/4$. Other terms follow similarly. Hence, we reach the ultimate expression for the forward recursion,

$$\begin{aligned}\alpha_i(j) = {} & \alpha_{i-1}(j)P_d^2 + \alpha_{i-1}(j-4)\frac{P_i^2}{16} + \alpha_{i-1}(j-2)\frac{P_iP_d}{2} \\ & + \alpha_{i-1}(j-1)P_tP_d\left(\sum_{x_{2i-1}} P(x_{2i-1})G(x_{2i-1}, y_{j-1}) + \sum_{x_{2i}} P(x_{2i})G(x_{2i}, y_j)\right) \\ & + \alpha_{i-1}(j-2)P_t^2\left(\sum_{x_{2i}} P(x_{2i})G(x_{2i}, y_j)\right)\left(\sum_{x_{2i-1}} P(x_{2i-1})G(x_{2i-1}, y_{j-1})\right) \\ & + \alpha_{i-1}(j-3)\frac{P_iP_t}{4}\left(\sum_{x_{2i}} P(x_{2i})G(x_{2i}, y_j) + \sum_{x_{2i-1}} P(x_{2i-1})G(x_{2i-1}, y_{j-2})\right),\end{aligned}$$

where,

$$G(x,y) = \begin{cases} 1 - P_s & \text{if } x = y, \\ P_s & \text{if } x \neq y. \end{cases}$$

Calculation of forward recursion term uses the marker information by exploiting perfect information from the probability term $P(x_i)$ for all x_i where,

$$P(x_i = 1) = \begin{cases} 1/2 & \text{if } i \text{ is not a marker index,} \\ 1 & \text{if } i \text{ is a marker marker index and } x_i = 1, \\ 0 & \text{if } i \text{ is a marker marker index and } x_i = 0. \end{cases}$$

Let us examine the derivation of the recursion terms for the backward probabilities, using a similar approach to that employed in the decomposition of the forward probabilities,

$$\begin{aligned}
\beta_i(j) &= P\left(\mathbf{y}_{j+1}^{R} \mid D_{i,j}\right) \\
&= \sum_{k=0}^{4} P\left(\mathbf{y}_{j+1}^{R} D_{i+1,j+k} \mid D_{i,j}\right) \\
&= \sum_{k=1}^{4} P\left(\mathbf{y}_{j+1}^{j+k}, \mathbf{y}_{j+k+1}^{R}, D_{i+1,j+k} \mid D_{i,j}\right) + P\left(\mathbf{y}_{j+1}^{R}, D_{i+1,j} \mid D_{i,j}\right) \\
&= \sum_{k=1}^{4} P\left(\mathbf{y}_{j+k+1}^{R} \mid D_{i+1,j+k}\right) P\left(\mathbf{y}_{j+1}^{j+k}, D_{i+1,j+k} \mid D_{i,j}\right) \\
&\quad + P\left(\mathbf{y}_{j+1}^{R} \mid D_{i+1,j}\right) P\left(D_{i+1,j} \mid D_{i,j}\right).
\end{aligned}$$

Recursion terms is obtained by realizing $P\left(\mathbf{y}_{j+1}^{R} \mid D_{i+1,j+k}\right) \triangleq \beta_{i+1}(j+k)$,

$$\begin{aligned}
\beta_i(j) &= \sum_{k=1}^{4} \beta_{i+1}(j+k) P\left(\mathbf{y}_{j+1}^{j+k}, D_{i+1,j+k} \mid D_{i,j}\right) + \beta_{i+1}(j) P\left(D_{i+1,j+k} \mid D_{i,j}\right) \\
&= \sum_{k=1}^{4} \beta_{i+1}(j+k) P\left(D_{i+1,j+k} \mid D_{i,j}\right) P\left(\mathbf{y}_{j+1}^{j+k} \mid D_{i+1,j+k}, D_{i,j}\right) \\
&\quad + \beta_{i+1}(j) P\left(D_{i+1,j+k} \mid D_{i,j}\right).
\end{aligned}$$

We again similarly decompose it as derived for the forward case and reach the ultimate expression for the backward recursion,

$$\begin{aligned}
\beta_i(j) = \beta_{i+1}(j) P_d^2 &+ \beta_{i+1}(j+4) \frac{P_i^2}{16} + \beta_{i+1}(j+2) \frac{P_i P_d}{2} \\
&+ \beta_{i+1}(j+1) P_t P_d \sum_{k \in \{1,2\}} \sum_{x_{2i+k}} P(x_{2i+k}) F(x_{2i+k}, y_{j+1}) \\
&+ \beta_{i+1}(j+2) P_t^2 \left(\sum_{x_{2i+1}} P(x_{2i+1}) F(x_{2i+1}, y_{j+1}) \right) \\
&\left(\sum_{x_{2i+2}} P(x_{2i+2}) F(x_{2i+2}, y_{j+2}) \right) \\
&+ \beta_{i+1}(j+3) \frac{P_i P_t}{4} \sum_{k \in \{1,2\}} \sum_{x_{2i+k}} P(x_{2i+k}) F(x_{2i+k}, y_{j+2k-1}).
\end{aligned}$$

At the last step, the likelihood terms, $P\left(\mathbf{y}_1^R \mid x_{2k-1}, x_{2k}\right)$ or $P\left(\mathbf{y}_1^R \mid \mathbf{s}_k\right)$ for $\mathbf{s}_k \in \{00, 01, 10, 11\}$ are calculated for $k \in \{1, \ldots, T/S\}$,

$$
\begin{aligned}
P\left(\mathbf{y}_1^R \mid \mathbf{s}_k\right) = P_d^2 & \sum_{n=0}^{\min(4k,R)} \alpha_{k-1}(n)\beta_k(n) \\
& + P_d P_t \sum_{n=0}^{\min(4k,R)} \sum_{i=0}^{1} \alpha_{k-1}(n-1)\beta_k(n)F(x_{2k-i}, y_n) \\
& + P_t^2 \sum_{n=0}^{\min(4k,R)} \alpha_{k-1}(n-2)\beta_k(n)F(x_{2k-1}, y_{n-1})F(x_{2k}, y_n) \\
& + \frac{P_i}{2} P_d \sum_{n=0}^{\min(4k,R)} \alpha_{k-1}(n-2)\beta_k(n) + \frac{P_i^2}{16} \sum_{n=0}^{\min(4k,R)} \alpha_{k-1}(n-4)\beta_k(n) \\
& + \frac{P_i}{4} P_t \sum_{n=0}^{\min(4k,R)} \sum_{i=0}^{1} \alpha_{k-1}(n-3)\beta_k(n)F(x_{2k-i}, y_{n-2i}).
\end{aligned}
$$

The initial conditions for the forward and backward equations are identical to those presented in Section 2.3. Additionally, the logarithmic domain versions of these equations employed in actual implementation, are similar to their bit-level counterparts.

As the number of bits comprising a symbol increases, the number of required terms for recursion and estimation expressions increases considerably. This makes the implementation of symbol-level MAP highly complicated for groupings of more than two or three bits.

4.3 Proposed Symbol Level Deep Learning Based Estimators

In this section, novel symbol-level deep learning-based estimators are proposed to enhance the performance of bit-level deep learning estimators and to decrease the overall complexity of symbol-level MAP detectors. We serially concatenate an LDPC with a marker code, and develop a BI-GRU architecture to estimate

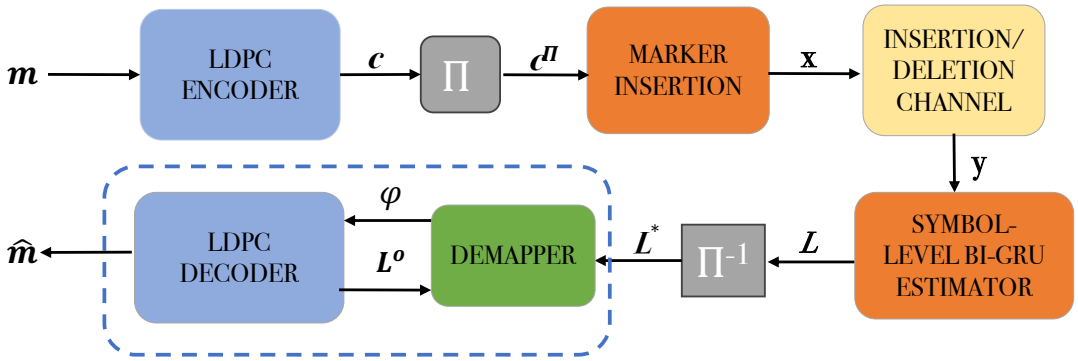

Figure 4.3: The block diagram for the proposed setup.

the posterior probabilities of the symbols input to the inner code. The aim of symbol-level BI-GRU is similar to that of the proposed method for the bit-level case despite the significant differences in architectural details. Fig. 4.3 illustrates the block diagram of the proposed setup.

Message vector $\mathbf{m} = (m_1, m_2 \ldots, m_k)$ of length k where $m_t \in \{0, 1\}$ is encoded through the outer (bit level) code, resulting in $\mathbf{c} = (c_1, \ldots, c_n)$ of length n where $c_t \in \{0, 1\}$. The rate of the outer code is k/n. Then, the encoded sequence $\mathbf{c}$ is grouped into symbols of length S and is expressed as $\mathbf{c} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{n/S})$, where each $\mathbf{s}_k = (c_{(k-1)S+1}, c_{(k-1)S+2}, \ldots, c_{kS})$ for $k \in \{1, 2, \ldots, n/S\}$. Each symbol is a group of S consecutive bits and can take values in $\{0, 1\}^S$. Since message vector, $\mathbf{m}$, is created randomly, $P(\mathbf{s}_k) = 1/2^S$ for all $\mathbf{s}_k \in \{0, \ldots, 2^S - 1\}$. The code length, n, may not be divisible by S. If that is the case, padding zeros until n divides S would solve the problem.

The encoded sequence $\mathbf{c} = (\mathbf{s}_1, \mathbf{s}_2 \ldots, \mathbf{s}_{n/S})$ is interleaved at the symbol-level, resulting in $\mathbf{c}^\pi$. Symbol-level interleaving permutes groups of bits rather than individual bits, and the interleaver is known to both the receiver and the transmitter. The interleaver is chosen as a random permutation. Then the pre-selected marker with length N_m is inserted after every N_c coded bit, resulting in the overall code, $\mathbf{x} = (x_1, \ldots, x_T)$ where T denotes the total number of transmitted bits, where $T = n + N_m(n/N_c)$.

The overall codeword is sent over the insertion and deletion channel and

$\mathbf{y}_1^R = (y_1, \ldots, y_R)$ is received where the length of the received sequence, R, is a random variable. The proposed BI-GRU-based estimator estimates the posterior probability term, $P(\mathbf{s}_k \mid \mathbf{y}_1^R)$ for every $\mathbf{s}_k \in \{0, \ldots, 2^S - 1\}$ and $k \in \{1, \ldots, n/Nc + n/S\}$. This estimator works slightly differently from the symbol-level MAP detector in the way that it directly estimates the posterior term without calculating the intermediate likelihood term, $P(\mathbf{y}_1^R \mid \mathbf{s}_k)$. Then, the S-dimensional vector at timestep k consisting of these posterior probabilities are formed, namely,

$$L_k = \left(P(\mathbf{s}_k = 0 \mid \mathbf{y}_1^R), P(\mathbf{s}_k = 1 \mid \mathbf{y}_1^R), \ldots, P(\mathbf{s}_k = 2^S - 1 \mid \mathbf{y}_1^R)\right).$$

For example, if $S = 2$,

$$L_k = \begin{bmatrix} P\left(\mathbf{s}_k = (0,0) \mid \mathbf{y}_1^R\right) \\ P\left(\mathbf{s}_k = (0,1) \mid \mathbf{y}_1^R\right) \\ P\left(\mathbf{s}_k = (1,0) \mid \mathbf{y}_1^R\right) \\ P\left(\mathbf{s}_k = (1,1) \mid \mathbf{y}_1^R\right) \end{bmatrix}.$$

After this calculation, the vector consisting of posterior probabilities are formed, $\mathbf{L} = (L_1, \ldots, L_{T/S})$. L_k's corresponding to marker vectors are removed since their values do not matter for the outer code decoder, resulting in the vector, $\mathbf{L} = (L_1, \ldots, L_{n/S})$. These values de-interleaved in the symbol level, resulting in the vector $\mathbf{L}^\pi = (L_1, \ldots, L_{n/S})$. $\mathbf{L}^\pi$ is input to a combination of the outer code and demapper module. There is a local information exchange between the demapper and the outer code decoder. After every N_o iteration of the outer code decoder, LLRs calculated by the outer code decoder, denoted by $\mathbf{L}^o = (L_1, \ldots, L_n)$, are input to demapper module. Demapper exploits the information outputted by the outer code decoder and MAP detector to update values, and its output $\Phi = (\varphi_1, \ldots, \varphi_n)$ is given to the outer code decoder. For $S = 2$, the demapper is

defined as follows, where $x_{2k-1} \in \{0,1\}$ and for $k \in \{1,\ldots,n/2\}$,

$$\begin{aligned}
\varphi(x_{2k-1}) &\triangleq P(\mathbf{y}_1^R \mid x_{2k-1}) \\
&= \sum_{\mathbf{s}_k} P(\mathbf{y}_1^R, \mathbf{s}_k \mid x_{2k-1}) \\
&= \sum_{\mathbf{s}_k} P(\mathbf{s}_k \mid x_{2k-1}) P(\mathbf{y}_1^R \mid \mathbf{s}_k, x_{2k-1}) \\
&= \sum_{\mathbf{s}_k} P(\mathbf{s}_k \mid x_{2k-1}) P(\mathbf{y}_1^R \mid \mathbf{s}_k) \\
&= \sum_{\mathbf{s}_k} \sum_{x_{2k} \in \{0,1\}} P(\mathbf{s}_k, x_{2k} \mid x_{2k-1}) P(\mathbf{y}_1^R \mid \mathbf{s}_k) \\
&= \sum_{\mathbf{s}_k} \sum_{x_{2k} \in \{0,1\}} P(x_{2k} \mid x_{2k-1}) P(\mathbf{s}_k \mid x_{2k}, x_{2k-1}) P(\mathbf{y}_1^R \mid \mathbf{s}_k),
\end{aligned}$$

where the term $P(\mathbf{y}_1^R \mid \mathbf{s}_k)$ is received from the symbol-level MAP estimator or symbol-level BI-GRU estimator, the term $P(x_{2k} \mid x_{2k-1})$ is received from the outer code decoder, and the term $P(\mathbf{s}_k \mid x_{2k-1}, x_{2k})$ is an indicator. Similarly, $\varphi(x_{2k})$ for $k \in \{1,\ldots,n/2\}$, is calculated as follows,

$$\varphi(x_{2k}) = \sum_{\mathbf{s}_k} \sum_{x_{2k} \in \{0,1\}} P(x_{2k-1} \mid x_{2k}) P(s_k \mid x_{2k}, x_{2k-1}) P(\mathbf{y}_1^R \mid \mathbf{s}_k).$$

Demapper for $S = 3$ repeats similar steps, for $k \in \{1,\ldots,n/3\}$, namely,

$$\begin{aligned}
\varphi(x_{3k}) &\triangleq P(\mathbf{y}_1^R \mid x_{3k}) \\
&= \sum_{\mathbf{s}_k} P(\mathbf{y}_1^R, \mathbf{s}_k \mid x_{3k}) \\
&= \sum_{\mathbf{s}_k} P(\mathbf{s}_k \mid x_{3k}) P(\mathbf{y}_1^R \mid \mathbf{s}_k, x_{3k}) \\
&= \sum_{\mathbf{s}_k} P(\mathbf{s}_k \mid x_{3k}) P(\mathbf{y}_1^R \mid \mathbf{s}_k) \\
&= \sum_{\mathbf{s}_k} \sum_{x_{3k-2}} \sum_{x_{3k-1}} P(\mathbf{s}_k, x_{3k-2}, x_{3k-1} \mid x_{3k}) P(\mathbf{y}_1^R \mid \mathbf{s}_k) \\
&= \sum_{\mathbf{s}_k} \sum_{x_{3k-2}} \sum_{x_{3k-1}} P(x_{3k-2} \mid x_{3k-1}, x_{3k}) P(x_{3k-1} \mid x_{3k}) \\
&\quad P(\mathbf{s}_k \mid x_{3k-2}, x_{3k-1}, x_{3k}) P(\mathbf{y}_1^R \mid \mathbf{s}_k).
\end{aligned}$$

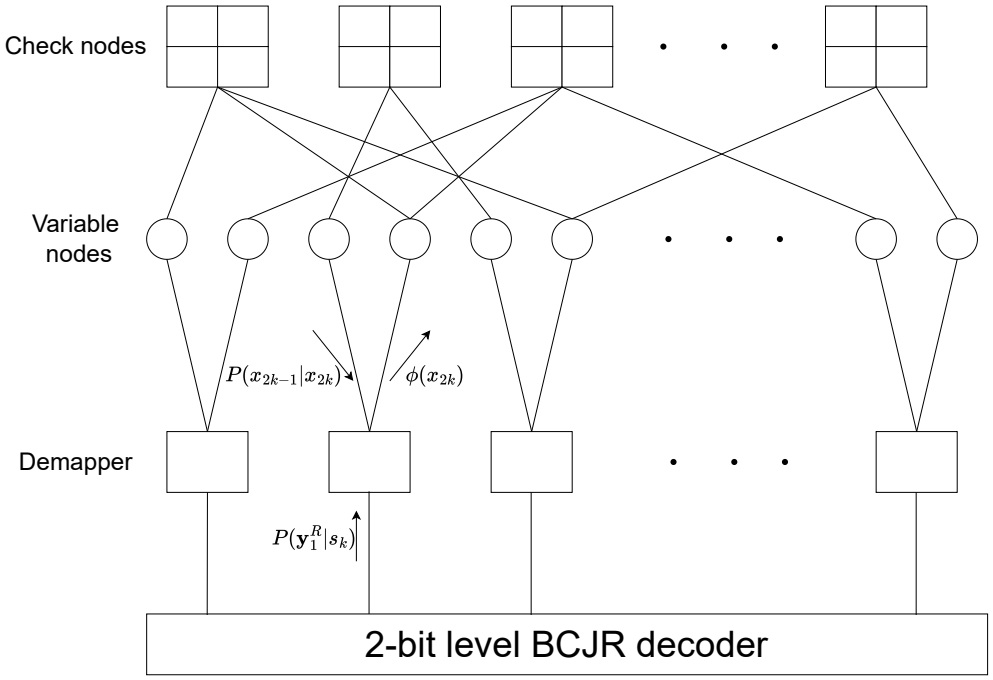

Figure 4.4: Illustration of the demapper module.

Fig. 4.4 shows an illustration of the demapper module for $S = 2$. After total N local exchanges between the demapper and the outer code decoder, the outer code decoder yields hard decisions on message bits, $\hat{\mathbf{m}} = (\hat{m}_1, \ldots, \hat{m}_k)$ where $\hat{m}_t \in \{0, 1\}$.

4.3.1 BI-GRU Estimators

In this section, symbol-level BI-GRU-based estimators are introduced. Symbol-level BI-GRU detectors estimate $P(\mathbf{s}_k \mid \mathbf{y}_1^R)$ for time steps $k \in \{1, \ldots, n/Nc + n/S\}$ and for every $\mathbf{s}_k \in \{0, \ldots, 2^S - 1\}$. We chose the BI-GRU architecture, similar to the reasons described in Chapter 3. That is why we employ with the BI-GRU architecture at the symbol level with little modifications in the architecture.

A BI-GRU architecture with N layers is used. The hidden state vectors of forward and backward BI-GRUs at time step t at last hidden layer (at state N) are $\overrightarrow{h}_t^{(N)}$, $\overleftarrow{h}_t^{(N)}$. On top of these vectors are applied a weight matrix of

W_o with a size $(2^S, 2d_{GRU})$. The output layer of the BI-GRU architecture at time step t in the symbol-level case outputs vector, $\mathbf{z}_t = W_o[\overrightarrow{h}_t^{(N)}; \overleftarrow{h}_t^{(N)}]$ for every $t = (1, \ldots, n/Nc + n/S)$ where $\mathbf{z}_t = (\mathbf{z}_{t,0}, \ldots, \mathbf{z}_{t,2^S-1})$ and $\mathbf{z}_{t,k} \in \mathbb{R}$. This vector is turned into a probability distribution by applying the softmax function, commonly used in multi-classification tasks in deep learning. Using $z_{t,k}$ for $k \in \{0, \ldots, 2^S - 1\}$ and applying softmax function, the posterior $P(\mathbf{s}_t = i \mid \mathbf{y}_1^R)$, for $t \in \{1, \ldots, n/Nc + n/S\}$, is calculated as follows,

$$P(\mathbf{s}_t = i \mid \mathbf{y}_1^R) = \frac{e^{z_{t,i}/\tau}}{\sum_{j=0}^{2^S-1} e^{z_{t,j}/\tau}}, \tag{4.1}$$

where τ is the temperature parameter to regulate the randomness of the distribution.

Another parameter γ is used to convert received vector $\mathbf{y}_1^R = (y_1, y_2, \ldots, y_R)$ into inputs to the BI-GRU detector at step t: The input to the BI-GRU detector at the step t is $\mathbf{X}_t = (y_{t-\gamma}, \ldots, y_t, \ldots, y_{t+\gamma})$ if t corresponds to the marker index otherwise, the input at time step t is the all-0 vector of length $2\gamma+1$. The marker index is where $t \in j\left(\frac{N_c}{S} + 1\right)$ for $j \in \{1, 2, \ldots, n/N_c\}$. This input representation makes sense since if we look at the backward and forward equations of the symbol-level MAP decoder; we observe that marker information utilized in the indexes corresponding to marker symbol indexes. We strictly require N_c to divide S. The expression for constructing inputs, for $t \in 1, \ldots, (n/Nc + n/S)$ is given as,

$$X_t = \begin{cases} (y_{t-\gamma}, \ldots, y_t, \ldots, y_{t+\gamma}) & \text{if } t = \left\{\left(\frac{N_c}{S} + 1\right), 2\left(\frac{N_c}{S} + 1\right), \ldots, \frac{n}{N_c}\left(\frac{N_c}{S} + 1\right)\right\}, \\ (0, 0, \ldots, 0) & \text{otherwise.} \end{cases} \tag{4.2}$$

Figure 4.5 shows the block diagram for symbol level BI-GRU estimator when $\gamma = 2$, $S = 2$, and $N_c = 2$.

4.3.2 Training Details

The BI-GRU estimators are trained without any fixed training set, meaning we have created each instance of input and output pairs during training; message

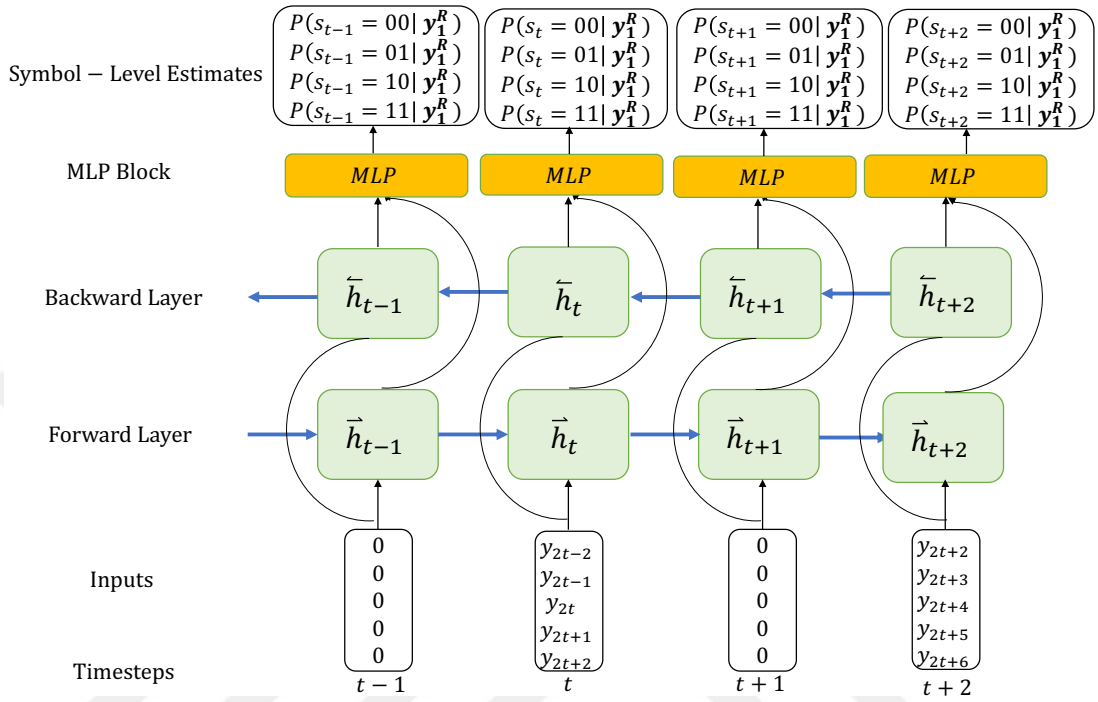

Figure 4.5: The block diagram for symbol level BI-GRU estimator when $\gamma = 2$, $S = 2$, and $N_c = 2$.

vector $\mathbf{m}$ is randomly created and encoded via the outer code and the inner code, resulting in the overall codeword to be sent through the channel, $\mathbf{x} = (x_1, \ldots, x_T)$. The output of the channel is $\mathbf{y}_1^R = (y_1, \ldots, y_R)$. Channel conditions should be chosen so that the network should generalize to a wide range of channel probabilities. To give an example, if we want to use the network for deletion probabilities $\{0.02, 0.03, 0.04\}$, then the network should be able to generalize for these probabilities of interest since we use the network for all channel conditions without any change. At the same time, we should be careful not to impede the training procedure by giving unreasonable probabilities.

An approach for determining suitable channel parameters for training is to determine the conditions where these networks are proposed to work $\mathbf{P}_d^{test} = (P_d^1, \ldots, P_d^{M_d})$, $\mathbf{P}_s^{test} = (P_s^1, \ldots, P_s^{M_s})$ and $\mathbf{P}_i^{test} = (P_i^1, \ldots, P_d^{M_i})$, where M_d, M_s, and M_i are the total number of test points for deletion, substitution and insertion probabilities, respectively. A random sample from all these probability terms is chosen for each training input. For example, if one decides to use the networks in

these channel conditions with $P_d \in (0.01, 0.02)$, $P_s = 0.05$, and $P_i \in (0.04, 0.1)$, then these are the training conditions for the channel. An important note is that models become unreliable for inference under testing conditions that fall outside the range of their training conditions. For instance, if a model is trained with channel conditions $P_d \in (0.01, 0.03)$, it may not perform well when tested with $P_d = 0.05$ unless the range of P_d is expanded to include this value as well.

Networks are trained with a supervised learning approach, and the task is multi-label classification of symbols. Cross-entropy loss is used as a loss function, where $\hat{y}_{t,b} = P(\mathbf{s}_t = b \mid \mathbf{y}_1^R)$ is the model prediction, and $y_{t,b}$ is the correct symbol transmitted through the channel, namely,

$$\mathcal{L}_{\text{CE}} = -\frac{1}{(n/Nc + n/S)} \sum_{t=1}^{(n/Nc+n/S)} \sum_{b=0}^{2^S-1} y_{t,b} \log(\hat{y}_{t,b}), \tag{4.3}$$

Since the indexes where markers are placed are trivial to calculate, the estimates of corresponding symbols, the loss function is not computed for those indexes. Consequently, this has no effect on the backpropagation process or the model parameters. We use the Adam optimizer [53] and a staircase learning decay [54], for which in each N_{wd} step of the Adam optimizer, the learning rate decays exponentially with parameter D during training. We use the default hyperparameters of the Adam optimizer. Batch normalization layers are added between the BI-GRU layers to train the network [55].

4.4 Numerical Results

In this section, we present some numerical results for symbol-level estimators. Table 4.1 shows the conditions under which the testing is performed, Table 4.2 shows the outer and inner code parameters, and Table 4.3 shows the hyperparameters of BI-GRU for these experiments. We adopted three different LDPC codes for the outer code with the following parameters: $(204, 102)$, $(504, 252)$, and $(816, 408)$. Testing at each point is terminated once a maximum of $30,000$

codewords or 200 frame errors is reached. We assume baseline methods fully know the channel's parameters.

	Channel Parameters		
	P_d	P_i	P_s
1,2	$(0.01, 0.025)$	0.00	0.01
3	$(0.01, 0.3)$	0.00	0.01
4	0.005	0.005	$(0.005, 0.025)$
5	0.1	$(0.01, 0.03)$	0.005
6,7	$(0.02, 0.04)$	0.00	0.00

Table 4.1: The specific insertion/deletion/substitution channels considered in the examples.

We trained all the networks for 30000 steps with the same learning rate 9×10^{-4} and employed staircase learning rate decay with decaying parameter 0.75. For a fair comparison, the model parameter sizes of all the BI-GRU symbol-level decoders for all S are equal in terms of N, d_{MLP}, and d_{GRU}.

	Inner Code		**Outer Code**
	Marker	N_c	(n, k)
1	$[0, 1]$	30	$(504, 252)$
2	$[0, 1, 0]$	20	$(504, 252)$
3	$[0, 1]$	12	$(204, 102)$
4	$[0, 1]$	12	$(204, 102)$
5	$[0, 1]$	12	$(204, 102)$
6	$[0, 1]$	12	$(204, 102)$
7	$[0, 1, 0]$	12	$(204, 102)$
8	$[0, 1]$	30	$(816, 408)$

Table 4.2: Inner and outer code parameters for the examples.

Fig. 4.6 or the first experiment compares the performance of symbol-level BI-GRU decoders for $S \in \{3, 5, 6\}$ with the baseline method for $S \in \{1, 2\}$ over a range of deletion probabilities for $P_s = 0.01$ and $P_i = 0$ (i.e., no insertions), and with marker code parameter $N_c = 30$. The outer code is the LDPC code with $n = 504$, $k = 252$. The figure shows that the frame error rate (FER) of trained networks for symbol bits $5, 6$ clearly is lower than baseline algorithms with $S = 2$. The bit error rate (BER) of trained networks is lower than the baseline algorithm for deletion probabilities $P_d \in \{0.01, 0.015\}$ and slightly higher

than that for $P_d \in \{0.02, 0.025\}$. As expected, our proposed models demonstrate consistency, as an increase in symbol length results in decreased FER and BER levels. This aligns with the assertion that performance improves for baseline and BI-GRU detectors as symbol length increases. Moreover, while the bit-level BI-GRU models could not outperform the bit-level baseline method, the 2-bit symbol-level BI-GRU network surpassed the bit-level baseline in FER and BER across all considered probabilities.

Fig. 4.7 or the second experiment compares the performance of symbol-level BI-GRU decoders for $S \in \{2, 4, 5\}$ with the baseline method for $S \in \{1, 2\}$ over a range of deletion probabilities, $P_d \in \{0.01, 0.015, 0.02, 0.025\}$, under the marker code parameter $N_c = 20$ and using the same outer code decoder as in the first experiment. In this case, the general trends observed in the first setup hold. However, this time, the BI-GRU decoders demonstrate a better performance in terms of BER. For FER, the results are similar, with the 5-bit BI-GRU decoder outperforming the 2-bit symbol-level baseline across all the deletion probabilities considered. We note that S must divide N_c, hence, when $N_c = 20$, BI-GRU models with $S \in \{2, 4, 5\}$ are developed, and when $N_c = 30$, models with $S \in \{3, 5, 6\}$ are developed.

	Model					**Training**		
	d_{GRU}	γ	d_{MLP}	N	S	lr	bs	D
1	256	40	128	2	$\{3, 5, 6\}$	9×10^{-4}	16	0.9
2	256	80	128	2	$\{2, 4, 5\}$	9×10^{-4}	16	0.9
3	256	40	128	2	$\{2, 3, 4\}$	9×10^{-4}	16	0.9
4	256	40	128	2	$\{2, 3, 4, 6\}$	9×10^{-4}	16	0.9
5	256	40	128	2	$\{3, 4\}$	9×10^{-4}	16	0.9
6,7	256	40	128	2	$\{2, 3, 4\}$	9×10^{-4}	16	0.9
8	256	40	128	2	$\{5, 6\}$	9×10^{-4}	16	0.9

Table 4.3: Configuration of BI-GRU model parameters for experimental setups.

We also conduct experiments using a shorter LDPC code with $n = 204$, $k = 102$. In this case, the BI-GRU decoders are trained for $S \in \{2, 3, 4\}$. Fig. 4.8 compares the performance of the symbol-level BI-GRU decoders with the baseline methods. For this code length, too, similar trends are observed: as the number of bits in each symbol increases, the performance curves of symbol-level

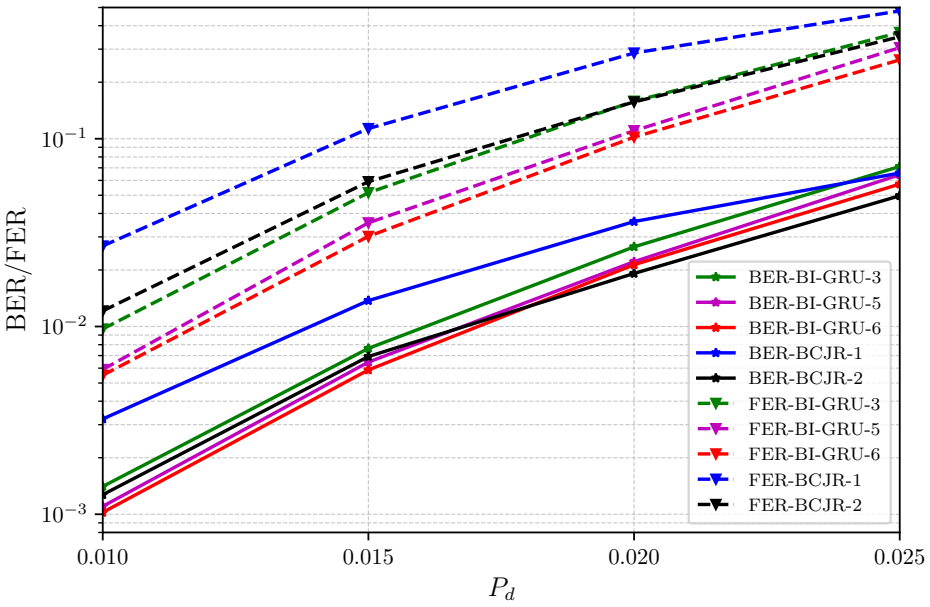

Figure 4.6: BER/FER as functions of deletion probability, when channel conditions are $P_s = 0.01$ and $P_i = 0$, for $S \in \{3, 5, 6\}$, $N_c = 30$, marker sequence $[0, 1]$, and outer LDPC code with $(416, 208)$.

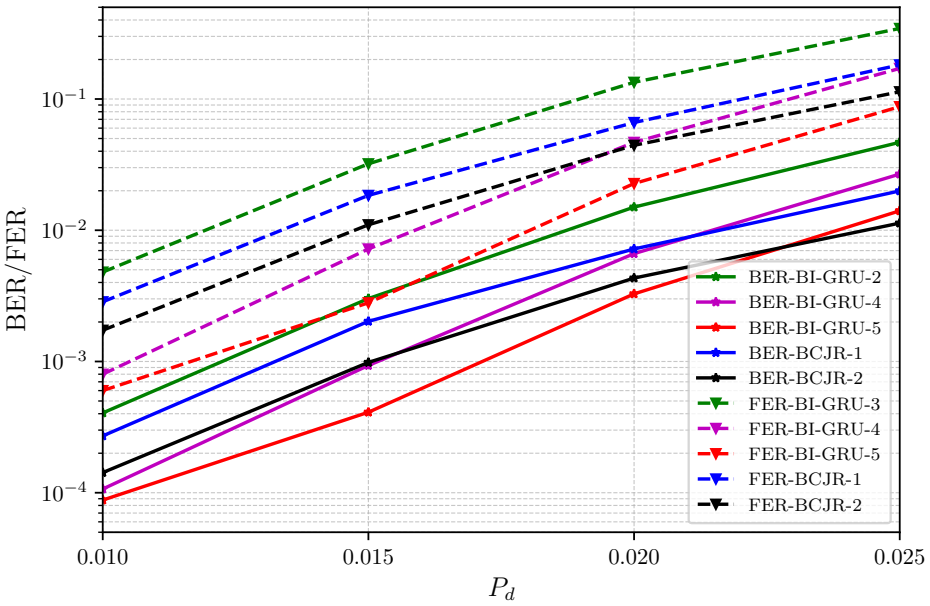

Figure 4.7: BER/FER as functions of deletion probability, when channel conditions are $P_s = 0.01$ and $P_i = 0$, for $S \in \{2, 4, 5\}$, $N_c = 20$, marker sequence $[0, 1, 0]$, and outer LDPC code with $(416, 208)$.

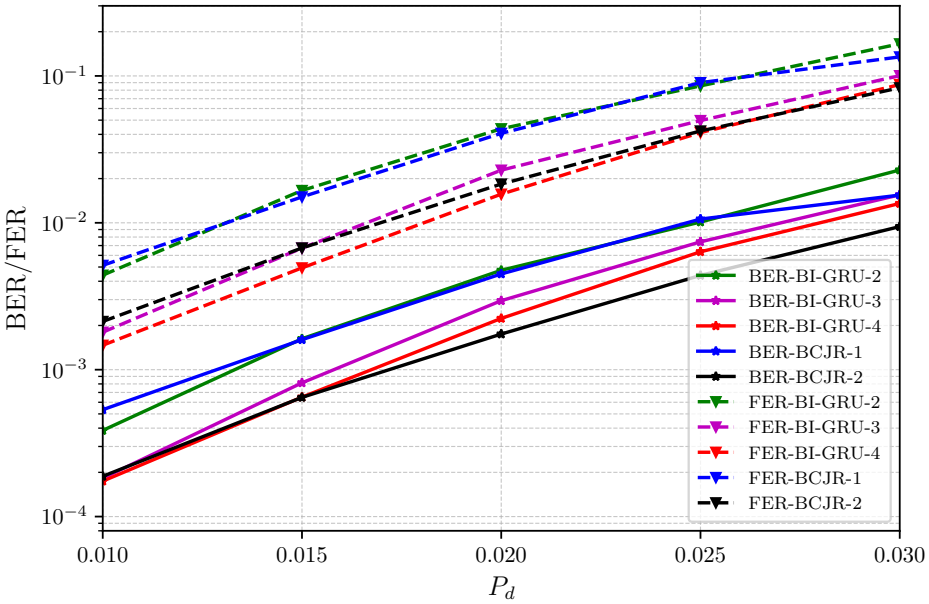

Figure 4.8: BER/FER as functions of deletion probability, when channel conditions are $P_s = 0.01$ and $P_i = 0$, for $S \in \{2, 3, 4\}$, $N_c = 12$, marker sequence $[0, 1]$ and, outer LDPC code with $(204, 102)$.

BI-GRU decoders improve in terms of both BER and FER. Specifically, the BI-GRU decoder with $S = 4$ for FER outperforms the baseline performance with $S = 2$. In terms of BER, an error rate of 2×10^{-4} is achieved when $P_d = 0.01$ for both BI-GRU-4 and BCJR-2. Additionally, we note that in this experiment, the P_d range is wider than in previous experiments, which impacts the training process.

Until now, we have considered the behaviors of the decoder for different deletion probabilities. We now conduct experiments to study the BI-GRU models' performance with varying P_i and P_s. Figs. 4.9 and 4.10 illustrate how symbol-level BI-GRU models perform for different values of P_s and P_i, respectively. The performance curves in Fig. 4.9, those for varying P_s, show that BI-GRU-3 and BI-GRU-4 outperform the bit-level baseline performance by a considerable margin in terms of both the BER and FER. Additionally, BI-GRU-4 and BI-GRU-6 surpass the 2-bit-level baseline performance, except at a substitution probability

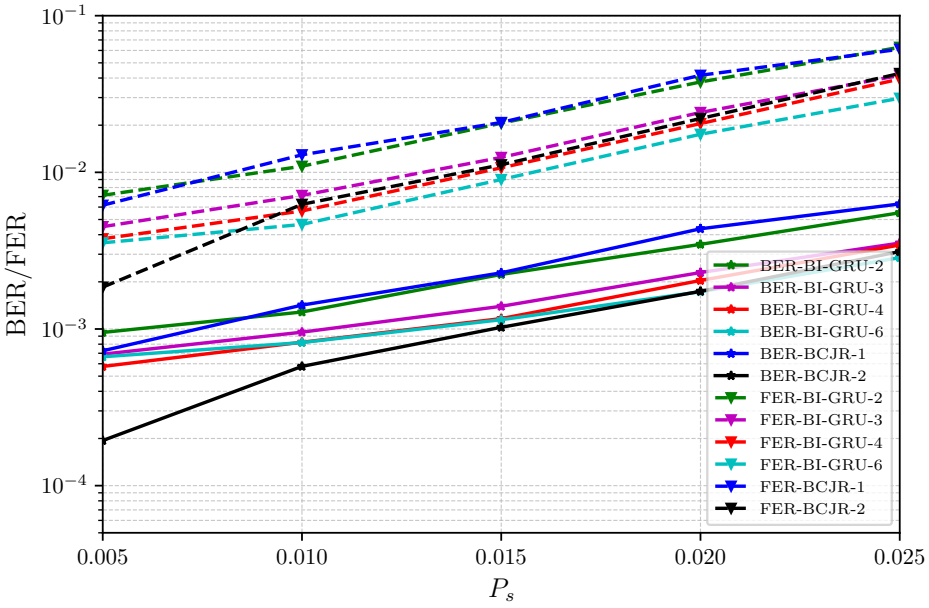

Figure 4.9: BER/FER as functions of substitution probability, when channel conditions are $P_s = 0.005$ and $P_i = 0.005$, for $S \in \{2, 3, 4, 6\}$, $N_c = 12$, marker sequence $[0, 1]$ and, outer LDPC code with $(204, 102)$.

of 5×10^{-3} for FER. This experiment highlights that symbol-level BI-GRU decoders can be trained and perform well not only for various deletion probabilities but also for varying substitution probabilities.

Similarly, the performance curves in Fig. 4.10, those for varying P_i, demonstrate that BI-GRU-3 and BI-GRU-4 outperform the bit-level baseline performance in terms of both the BER and FER. BI-GRU-4 also outperforms the 2-bit-level baseline performance for insertion probabilities of 0.01 and 0.02, but it remains inferior at $P_i = 0.03$.

The final experiments are only for deletion channels without substitutions or insertions. Figs. 4.11 and 4.12 show the error curves for deletion probabilities of 0.02, 0.03, and 0.04. Fig. 4.13 is an experiment with a longer LDPC code, with $n = 816$, $k = 408$.

The above results demonstrate that the symbol-level decoders provide solutions for number of bits in each symbol up to 6, exceeding the threshold for the

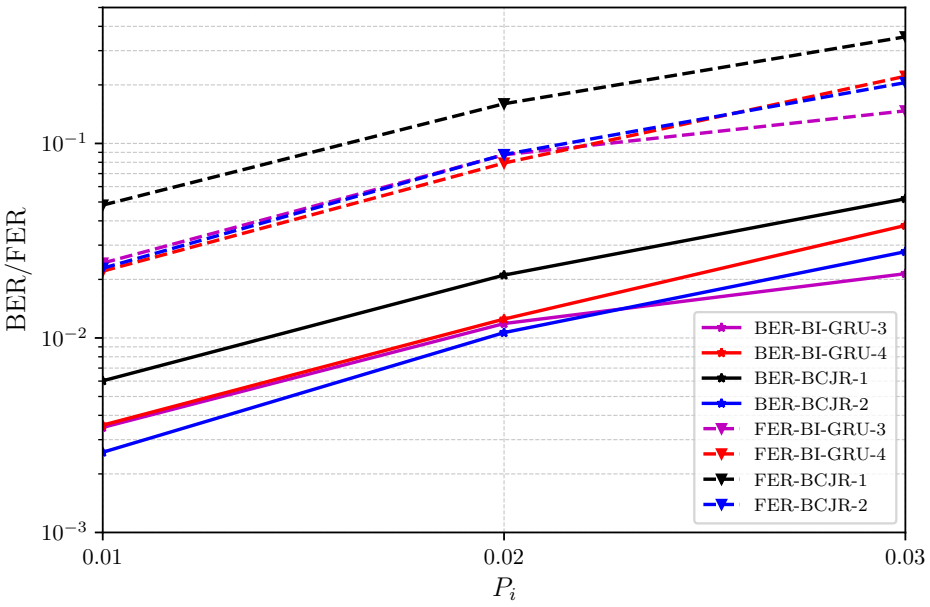

Figure 4.10: BER/FER as functions of insertion probability, when channel conditions are $P_d = 0.01$ and $P_s = 0.005$, for $S \in \{3, 4\}$, $N_c = 12$, marker sequence $[0, 1]$, and, outer LDPC code with $(204, 102)$.

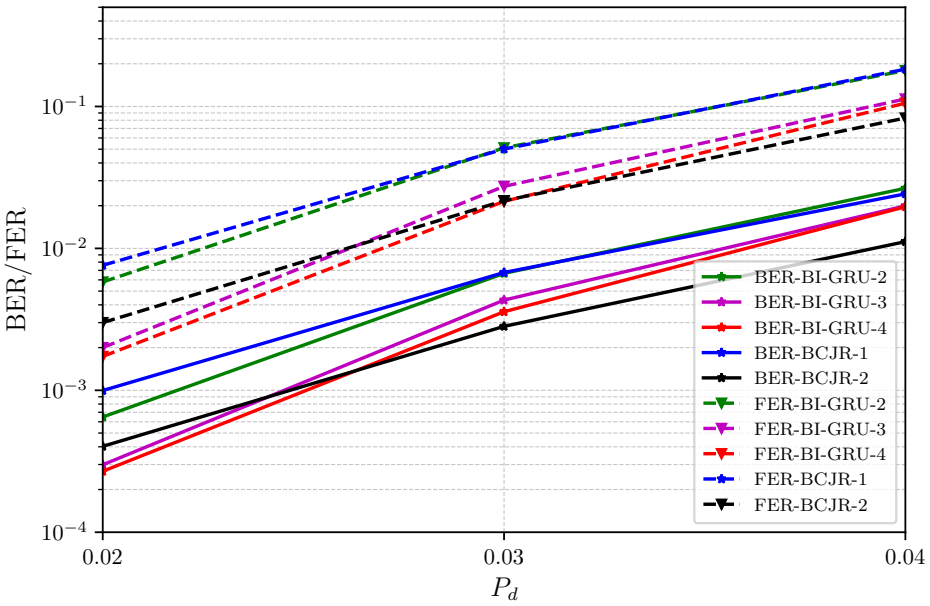

Figure 4.11: BER/FER as functions of deletion probability, when channel conditions are $P_s = 0.01$ and $P_i = 0$, for $S \in \{2, 3, 4\}$, $N_c = 12$, marker sequence $[0, 1]$, and outer LDPC code with $(204, 102)$.

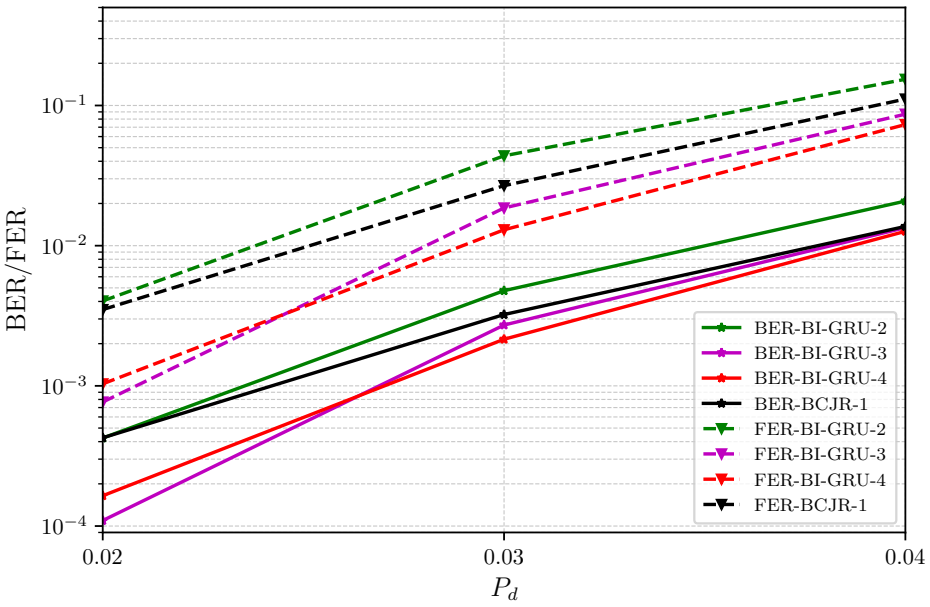

Figure 4.12: BER/FER as functions of deletion probability, when channel conditions are $P_s = 0.01$ and $P_i = 0$, for $S \in \{2, 3, 4\}$, $N_c = 12$, marker sequence $[0, 1, 0]$, and outer LDPC code with $(204, 102)$.

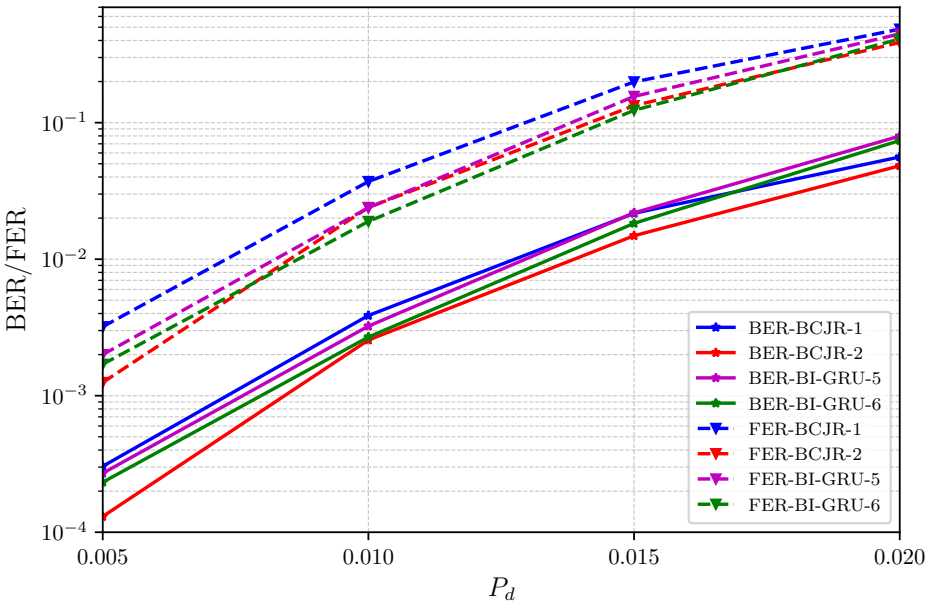

Figure 4.13: BER/FER as functions of deletion probability, when channel conditions are $P_s = 0.01$ and $P_i = 0$, for $S \in \{5, 6\}$, $N_c = 30$, marker sequence $[0, 1]$ and outer LDPC code with $(816, 408)$.

baseline symbol-level decoders (which were limited to $S \leq 3$). Furthermore, we observe that BI-GRU decoders exhibit strong performance across various probability ranges.

Increasing S in deep learning decoders beyond some point presents significant challenges as well, as in symbol-level MAP detectors. As S increases, the number of classes in the final layer of the symbol-level BI-GRU detector increases exponentially to 2^S. While theoretically, performance improves with a higher S, practical constraints in deep learning models limit this growth.

4.5 Complexity Analysis

At each time instance k, where $k \in \{1, \ldots, T/S\}$, we have approximately $20Nd_{GRU}^2$ multiplications and additions (same as the bit-level decoders for each time step). For all time instances, with a total of T/S time steps, there are approximately $12N\frac{T}{S}d_{GRU}^2$ multiplications and additions.

Similar arguments can be made to those in Section 3.4. As T increases due to the use of a large code with a large n, the complexities of both methods become comparable for small values of n. However, for larger n, the complexity of symbol-level BI-GRU can be reduced significantly compared to that of symbol-level MAP algorithm.

4.6 Chapter Summary

This chapter proposes BI-GRU-based estimators as a symbol-level detection method for concatenated codes, extending bit-level BI-GRU-based estimators over the insertion/deletion channel. Proposed symbol level BI-GRU decoders improved error rate performance of baseline methods employing bit-level decoders,

including BI-GRU bit-level decoders and baseline bit-level MAP detectors. Additionally, symbol-level decoders were capable of processing symbols with more than three bits and performing calculations where baseline methods failed. While the baseline methods became intractable for symbols larger than three bits, deep learning-based methods handled symbols up to six bits. Symbol-level decoders provide a competitive alternative solution in terms of both FER and BER.

Chapter 5

Deep Learning Decoders for Insertion and Deletion Channels with ISI

In this chapter, the bit-level bi-directional gated recurrent unit (BI-GRU) decoders proposed in Chapter 4 are adapted to estimate bits encoded by the outer code in concatenated codes over insertion and deletion channels, further complicated by intersymbol interference (ISI). We demonstrate that deep learning-based estimators provide effective solutions in scenarios where deriving explicit analytical expressions is difficult, and implementation complexities are too high. A key objective of this chapter is to extend the research findings to insertion and deletion channels with ISI, encountered in practical applications like bit-patterned media recording and nanopore sequencing, emphasizing the potential of deep learning-based decoders in more complex settings.

In some channel models, ISI coexists with insertions and deletions. For instance, the authors in [12] consider synchronization errors and ISI in bit-patterned media recording systems, leading to insertion/deletion channels with ISI. This channel model can be considered as a cascade of an insertion and deletion channel followed by an ISI channel. Another related scenario is the nanopore sequencing

channel [13], which also deals with ISI. Nanopore sequencing involves encoding digital data into DNA strands and passing these sequences through a nanoscale biological membrane. As nucleotides pass through the membrane, they generate current levels that the sequencer uses to identify each nucleotide. However, these current levels are affected by both the current and previous nucleotides, leading to ISI from earlier bits. We adopt the model in [12]; however, the general ideas can be employed for the model in [13] as well.

This chapter is organized as follows: Section 5.1 introduces the channel model. Section 5.2 briefly reviews the related literature. In Section 5.3, we propose a novel symbol-level BI-GRU approach for addressing insertion and deletion channels with ISI. Section 5.4 presents several numerical results, and the chapter concludes with a summary in Section 5.5.

5.1 Channel Model

We consider an insertion/deletion channel with ISI. This channel is a cascade of insertion and deletion channels with probabilities P_i, P_d, and P_s, denoting insertion, deletion, and substitution probability, respectively, (described in Section 4.1) and an ISI channel. The codeword sequence, $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ where each bit $x_i \in \{+1, -1\}$, is sent first through the insertion and deletion channel. Due to random insertions and deletions, this sequence is transformed into a new sequence $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{\tilde{R}})$ where each bit $\tilde{x}_i \in \{+1, -1\}$. This sequence is then transmitted through an ISI channel characterized by the channel taps $\mathbf{h} = (h_0, \ldots, h_L)$, where the number of channel taps is $L + 1$ and $h_i \in \mathbb{R}$. Additionally, the transmitted signal is affected by additive white Gaussian noise (AWGN) z_k, which has a mean of zero and a variance of $\sigma^2 = \frac{N_0}{2}$. The signal-to-noise ratio (SNR) is defined as $\text{SNR} = \frac{1}{N_0}$. The input-output relationship is given by:

$$y_k = \sum_{l=0}^{L} h_l \tilde{x}_{k-l} + z_k,$$

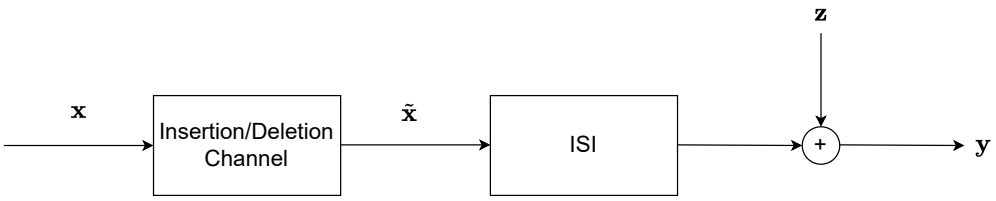

Figure 5.1: Cascade of an insertion/deletion channel with an ISI.

for $k \in \{1, 2, \ldots, R\}$. The channel taps $\mathbf{h}$ are normalized such that $\sum_{l=0}^{L} |h_l|^2 = 1$. The received sequence is denoted by $\mathbf{y}_1^R = (y_1, \ldots, y_R)$. The overall channel model is illustrated in Fig. 5.1.

5.2 Information Rates of Insertion/Deletion Channels with ISI

This section reviews the information rates of insertion/deletion channels with ISI determined in the previous literature. The method proposed in [12] combines the channel states due to ISI and those due to insertions and deletions (the states corresponding to the insertion and deletions in the BCJR algorithm explained in Sections 2.3 and 4.2) to obtain a joint trellis representation. The trellis representation considers the states due to insertion and deletions, and ISI. The event is defined as $D_k = (i, j)$, a length-2 vector, where i represents the number of deletions (or insertions) up to the transmission of the i^{th} bit x_i ($i \in \{1, 2, 3, \ldots\}$), and j denotes the current channel state. For example, if n insertions ($n = 0, 1, \ldots$) occur between x_{k-1} and x_k, the state transitions from $D_{k-1} = (i, j)$ to $D_k = (i + n, j_2)$, with the corresponding output segment y_{k+i+n}^{k+i} used in branch calculations. Here, j_2 is determined probabilistically based on the channel state at time $k - 1$ and the input x_k.

This trellis representation can be used for detecting the inner marker codes and producing soft information about the bits (encoded by the outer code) in a concatenated coding setup. This is similar to the baseline methods adopted in

the previous two chapters for the case with no ISI. However, the trellis complexity may be too high due to the presence of ISI along with the insertions and deletions. Therefore, this method is not very practical in terms of implementation.

In [12], the trellis representation has been used to estimate achievable information rates over insertion/deletion channels with ISI using independent and identically distributed (i.i.d.) channel inputs. The idea is as follows. A long (simulated) codeword $\mathbf{x}_1^T = (x_1, \ldots, x_T)$, where the elements are i.i.d., is transmitted through an insertion/deletion channel with ISI, and $\mathbf{y}_1^R = (y_1, \ldots, y_R)$ is obtained. Then using the forward recursion of the BCJR algorithm, the quantity $p(\mathbf{x}_1^T, \mathbf{y}_1^R)$ is estimated, which is then employed to generate an estimate of the mutual information over the channel. Using long sequences, and if needed, by averaging over multiple realizations, it is possible to obtain the achievable rates with any desired accuracy for smaller insertion/deletion probabilities. It is also possible to extend this method to Markov inputs and obtain achievable rates.

As a remedy to the complexity of the standard BCJR type detection over insertion/deletion channels with ISI, deep learning based channel detectors can be developed, which is the subject of this chapter. Indeed, it is possible to even employ more powerful symbol-level detectors for this complicated setup, as will be illustrated in the rest of the chapter.

5.3 Deep Learning Based Estimators for Insertion/Deletion Channels with ISI

5.3.1 System Model

In this section, we propose novel symbol-level deep learning-based estimators for concatenated codes over insertions and deletion channels with ISI. Fig. 5.2 illustrates the block diagram of the proposed setup.

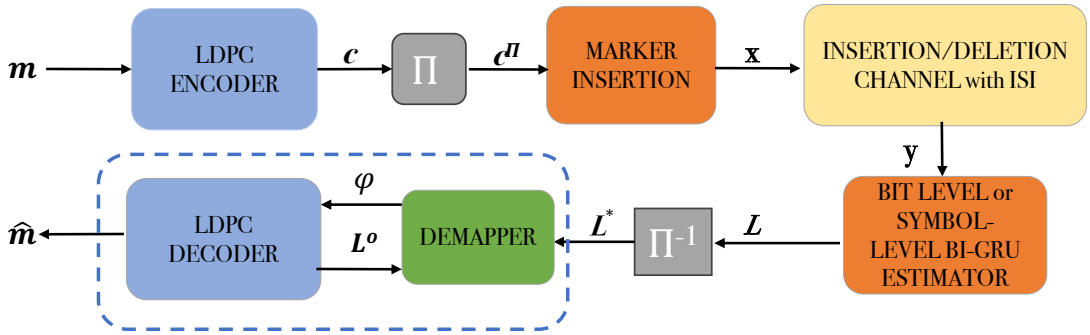

Figure 5.2: The block diagram for the proposed setup.

The message vector $\mathbf{m} = (m_1, \ldots, m_k)$ where $m_k \in \{-1, +1\}$ is encoded by the outer LDPC code and the inner marker code. The encoding is the same as those of the system models proposed in Chapter 4, however, the bit 0 is mapped to -1 and the bit 1 is mapped to -1. The transmitted sequence of length T is denoted by $\mathbf{x}_1^T = (x_1, \ldots, x_T)$, where $x_k \in \{-1, +1\}$, and the received sequence by $\mathbf{y}_1^R = (y_1, \ldots, y_R)$, where $y_k \in \mathbb{R}$, which is affected not only by insertions and deletions but also by ISI and AWGN noise. The proposed BI-GRU-based estimator computes the probability term $P(\mathbf{s}_k \mid \mathbf{y}_1^R)$ for each $\mathbf{s}_k \in \{0, \ldots, 2^S - 1\}$ and $k \in \{1, \ldots, n/N_c + n/S\}$ at the symbol level, forming $\mathbf{L} = (L_1, \ldots, L_{T/S})$. These values are deinterleaved at the symbol level, and the resulting sequence $\mathbf{L}^\pi$ is passed to a combination of the outer code and demapper module. The calculations follow the same procedure as in Chapter 4, with the key difference that the received sequence $\mathbf{y}_1^R$ consists of real-valued elements, and they experience ISI from neighboring symbols as well as AWGN.

5.3.2 BI-GRU Estimators

We use symbol-level decoders to calculate the posterior probabilities because they provide better error rate performance compared to bit-level decoders, as demonstrated by the experimental results in Chapter 4. The same symbol-level BI-GRU estimator from Chapter 4 is employed, retaining the same architectural details. The symbol-level BI-GRU estimators estimate $P(\mathbf{s}_k \mid \mathbf{y}_1^R)$ for time steps

$k \in \{1, \ldots, n/N_c + n/S\}$ and for each $s_k \in \{0, \ldots, 2^S - 1\}$, where the received sequence elements $y_k \in \mathbb{R}$ for $k \in \{1, \ldots, R\}$, which is different from the models trained for just insertion and deletion channels where $y_k \in \{0, 1\}$.

The other difference between symbol-level models developed in Chapter 4 and those trained for the insertion/deletion channels with ISI is in its input representation defined by the parameter γ. γ denotes the run length of the inputs at timestep t, as defined in Section 4.3. Particularly, the input at the time step t is $x_t = (y_{t-\gamma}, y_{t-\gamma+1}, \ldots, y_t, \ldots, y_{t+\gamma-1}, y_{t+\gamma})$ for $t \in \{1, \ldots, n/Nc + n/S\}$. The label in each time step is $\mathbf{s}_k$.

5.3.3 Training Details

The networks in this chapter are trained without a fixed training set. Specifically, a message vector $\mathbf{m} = (m_1, \ldots, m_k)$ is generated randomly and encoded using the outer LDPC code and the inner marker code, resulting in the codeword $\mathbf{x} = (\mathbf{s}_1, \ldots, \mathbf{s}_{n/N_c+n/S}) = (x_1, \ldots, x_T)$, where $\mathbf{s}_k$ represents the symbol grouping. The codeword $\mathbf{x}$ is then transmitted over an insertion/deletion channel with ISI, characterized by the parameters P_d, P_i, P_s, and σ. The received sequence is denoted as $\mathbf{y}_1^R$, where $\mathbf{x}$ serves as the label and $\mathbf{y}_1^R$ serves as the input to the network. We give inputs and labels in batches.

As with the models in Chapter 4, the channel conditions for training are critical and should be chosen carefully. The parameter σ determines the SNR of the channel, and different SNR values are provided during training to enable a robust performance. A random sample from these probability terms is selected for each training input, and the channel is modified accordingly. To prevent instabilities during training, gradient clipping is employed, which limits the gradients to a pre-determined norm value.

The networks are trained using a supervised learning approach, with cross-entropy loss as the objective function. The Adam optimizer is used, and a staircase learning rate decay is applied.

5.4 Numerical Examples

In this section, we provide numerical results for BI-GRU based receivers for concatenated codes over insertion/deletion channels with ISI. A commonly used channel coefficients in recording systems is the partial response-4 (PR4) target, which is used throughout all the experiments in this section, whose channel response is $\mathbf{h} = \left[\frac{1}{\sqrt{2}},\ 0,\ -\frac{1}{\sqrt{2}}\right]$ [4]. The model sizes are provided in Table 5.1, showing that they are larger than the BI-GRU models designed for insertion and deletion channels without ISI, as listed in Table 4.3. The increase in model size is due to the addition of two more layers (for a total of four) and an expansion of the hidden vector dimensions to 512 units. The previous configuration in Chapter 4 had two layers and 256 hidden units. This adjustment is necessary because of the challenging nature of the channel, which is influenced by ISI channel taps and the presence of Gaussian noise. Gradient clipping with a norm of 1 is applied to control gradient magnitudes. Training the BI-GRU models for these channels is significantly more difficult than for the models in the previous chapter, requiring a parameter search for the learning rate and batch size. The parameters used for training include $N_c \in \{6, 12\}$, $\gamma = 3$, and a marker sequence of $[0, 1, 0]$. The networks were trained for 30,000 steps.

Fig. 5.3 shows the performance of the proposed decoder for a deletion channel with $P_d = 0.01$ for different values of the parameter $\gamma \in \{5, 10, 20, 40\}$. The x-axis represents the E_b/N_0, which differs from the earlier figures where the x-axis typically represented the range of deletion or insertion probabilities, or other channel parameters. These experiments were conducted to select the best γ parameter among different possibilities. As observed, the model with $\gamma = 40$ outperforms the others in terms of bit error rate (BER) and frame error rate (FER), and thus this value is adopted in the subsequent settings.

Fig. 5.4 presents the performance of the proposed decoder under different channel conditions. The first scenario is the deletion channel with $P_d = 0.01$, and the second scenario involves a channel with both substitutions and deletions with $P_d = 0.01$ and $P_s = 0.01$. We employ symbol-level deep learning architectures

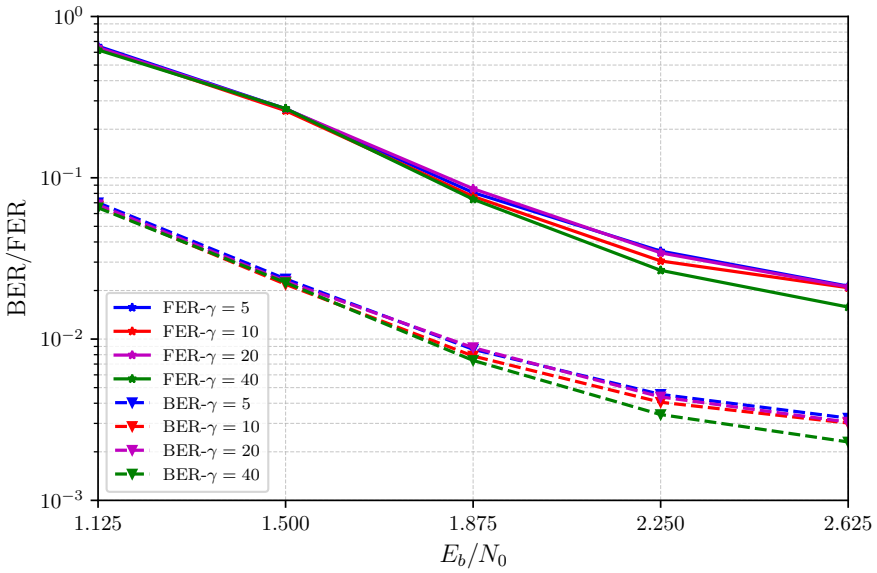

Figure 5.3: First Experiment, BER/FER as functions of E_b/N_0 and γ, when channel conditions $P_d = 0.01$ and $P_s = 0.00$, for $S = 2$, $N_c = 6$, marker sequence $[0, 1, 0]$, and outer LDPC code with $(204, 102)$.

with two bits per symbol ($S = 2$). As expected, the model performs better under more favorable channel conditions, with the deletion-only channel with $P_d = 0.01$ outperforming the channel having $P_d = 0.01$ and $P_s = 0.01$ for all the E_b/N_0 values considered. A BER of 10^{-2} is achieved at an E_b/N_0 of 2.25 dB for the BI-GRU decoder on the channel with $P_d = 0.01$ and $P_s = 0.01$, and at an E_b/N_0 of 1.7 dB for the BI-GRU decoder over the deletion channel with $P_d = 0.01$.

	Model					**Training**		
	d_{GRU}	γ	d_{MLP}	N	S	lr	bs	N_c
1	512	$\{5, 10, 20, 40\}$	128	4	2	9×10^{-4}	16	6
2	512	40	128	4	2	9×10^{-4}	16	6
3	512	40	128	4	2	9×10^{-4}	16	6
4	512	40	128	4	2	9×10^{-4}	16	$\{6, 12\}$

Table 5.1: Configuration of BI-GRU model parameters for experimental setups.

Fig. 5.5 shows the performance of the proposed decoder for two different deletion probabilities $P_d = 0.01$, and $P_d = 0.02$, while fixing the substitution probability at $P_s = 0.01$. As expected, the performance of the model for more favorable channel conditions performs better for all the E_b/N_0 values considered.

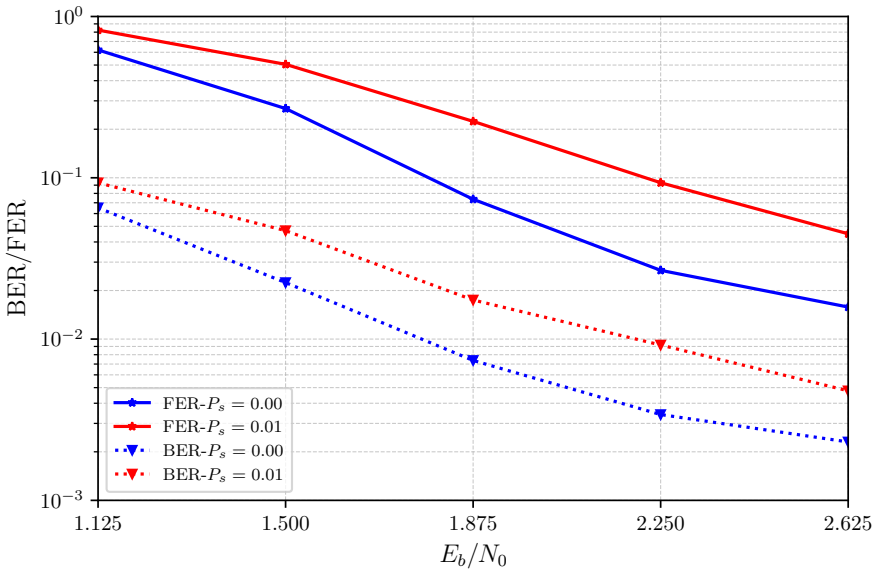

Figure 5.4: Second Experiment, BER/FER as functions of E_b/N_0, when channel conditions $P_d \in 0.01$ and $P_s \in \{0.00, 0.01\}$, for $S = 2$, $N_c = 6$, marker sequence $[0, 1, 0]$, and, outer LDPC code with $(204, 102)$.

A BER of 5×10^{-2} is achieved at an E_b/N_0 of 2.05 dB for BI-GRU based receiver for the channel with $P_d = 0.01$ and $P_s = 0.01$; while an E_b/N_0 of 2.25 dB is needed for the channel with $P_d = 0.02$ and $P_s = 0.02$.

Lastly, Fig. 5.6 shows the performance of the proposed decoder for $N_c = 6$ and $N_c = 12$ under the same channel conditions for both models, i.e., for a deletion only channel with $P_d = 0.01$.

The numerical results demonstrate the powerful decoding capabilities of BI-GRU-based detectors for concatenated codes over insertion and deletion channels with ISI, a scenario that has proven challenging. Although the existing methods discussed in Section 5.2 for this channel can be adapted to the concatenated code setting where inner codes serve as markers, implementing maximum a posteriori (MAP) detection for these channels remains highly complex, even at the bit level. The results presented here may pave the way for applying deep learning-based estimators to other variations of insertion and deletion channels further degraded by ISI.

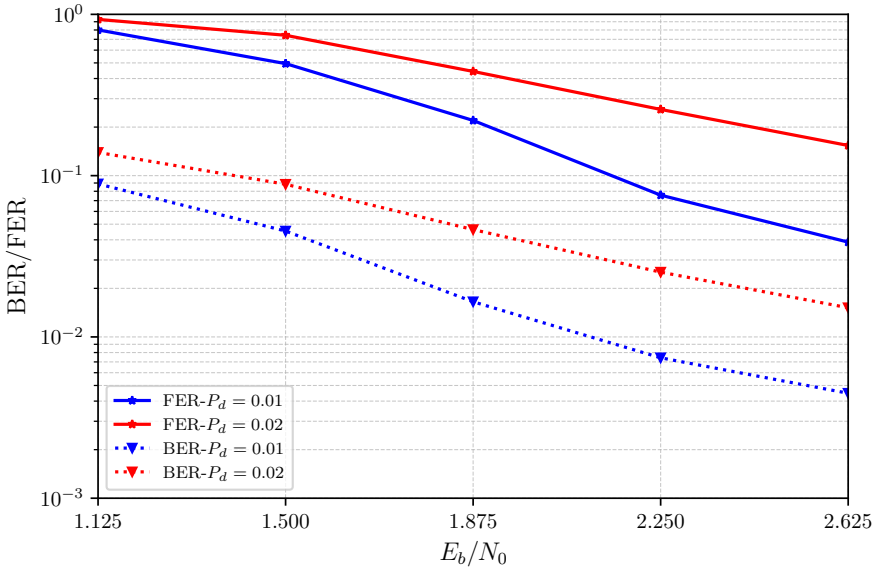

Figure 5.5: Third Experiment, BER/FER as functions of E_b/N_0, when channel conditions $P_s = 0.01$ and $P_d \in \{0.01, 0.02\}$, for $S = 2$, $N_c = 6$, marker sequence $[0, 1, 0]$ and, outer LDPC code with $(204, 102)$.

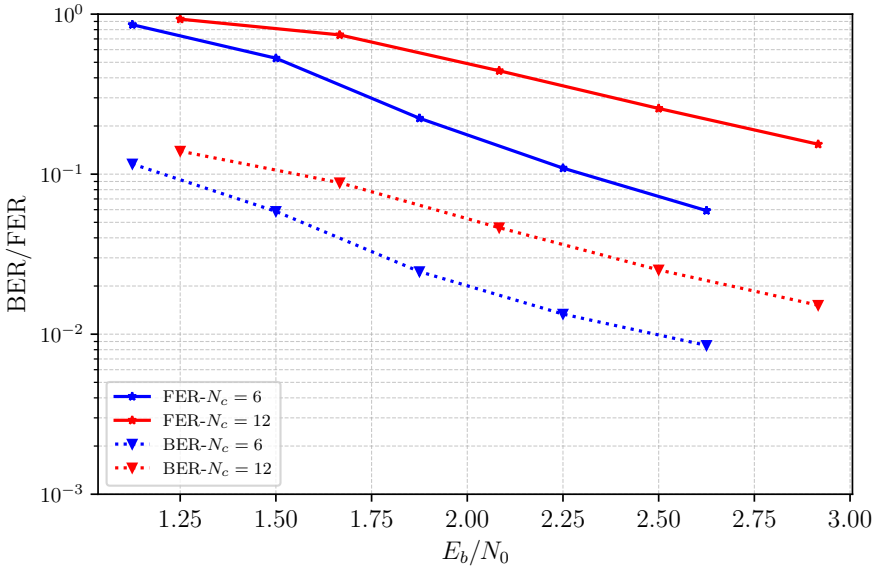

Figure 5.6: Forth Experiment, BER/FER as functions of E_b/N_0, when channel condition $P_d = 0.01$, for $S = 2$, $N_c \in \{6, 12\}$, marker sequence $[0, 1, 0]$ and, outer LDPC code with $(204, 102)$.

5.5 Chapter Summary

In this chapter, we proposed symbol-level BI-GRU decoders for concatenated codes over insertion and deletion channels degraded by ISI. This is an important model for certain magnetic recording channels. The results demonstrate that the BI-GRU decoders introduced in earlier chapters can be adapted to more challenging scenarios, including the insertion and deletion channels with ISI. Furthermore, the BI-GRU-based models offer low-complexity decoding solutions, for scenarios where the standard BCJR-algorithm based methods become impractical due to the significantly increased number of states in the trellis diagram.

Chapter 6

Conclusions and Future Directions

This thesis presents deep learning based decoders for concatenated codes, where the inner code is a marker code, applied over channels with synchronization errors at both the symbol and bit levels. The primary motivation behind this work is to provide an alternative decoding method for estimating the log-likelihood ratios (LLRs) of the bits encoded by the outer code, which are then employed by the outer code's decoder to complete the decoding operation.

We propose bit-level deep learning-based estimators for concatenated codes where the outer code is an low-density parity-check (LDPC) code and the inner code is a marker code. We also develop a one-shot decoder for serial concatenation of convolutional and marker codes via a complete BI-GRU-based architecture, directly estimating message bits. We employ BI-GRU-based estimators instead of other deep learning-based decoders because they can handle variable-length inputs, which are common in channels that alter the size of the input sequence. BI-GRU models also resemble the forward and backward algorithm, which are commonly used in traditional decoding algorithms for concatenated codes.

The traditional BCJR algorithm requires channel parameters to estimate bit

LLRs and in this case, the algorithm becomes suboptimal. On the contrary, when channel conditions are not fully known to the receiver, bit-level-based models have shown excellent performance across a broad range of probabilities. One additional advantage of bit-level decoders is their ability to operate for different channel parameters without requiring structural changes or additional inputs.

To further enhance the error rate performance, we also propose deep-learning based symbol-level decoders, which outperform the baseline methods based on bit-level decoders. Symbol-level decoders exploit the correlations among neighboring bits that are affected by insertion/deletion channels, leading to improved performance. However, limitations arise in the baseline symbol-level maximum a posteriori (MAP) decoder, which can group up to three bits due to increased complexity. In contrast, deep-learning based symbol-level decoders can handle longer symbols exceeding this threshold, that is, they can be used when the common baseline technique is not feasible.

We have conducted experiments with insertion and deletion channels further degraded by inter-symbol interference (ISI). Our results demonstrate that deep learning-based methods can estimate LLRs for these challenging models, whereas traditional algorithms are too complex due to the increase in the number of states in the trellis representation. These findings suggest that deep learning techniques offer a promising alternative to traditional methods for decoding over complicated channel models with synchronization errors, ISI, and potentially other impairments.

Future research directions are manifold. For instance, one can use pruning and quantization techniques to decrease network size and inference times. This objective is important to further decrease the complexity of the decoder architectures.

Another possible research direction is extension of the proposed architectures to non-binary alphabets. A primary application of such an extension is in DNA

storage, where the unique nature of these channels presents an immediate opportunity. Various channel models exist for DNA storage systems, such as noisy-shuffling channels, and nanopore sequencing. Deep learning models can potentially be adapted to address issues specific to these applications. For instance, they may be designed to correct the synchronization errors and restore order in noisy-shuffling channel models. Additionally, in nanopore channels, the deep learning based decoders may be used to mitigate the effects of ISI along with the substitution errors.

It may also be possible to use deep learning techniques to calculate capacity bounds for insertion/deletion channels. Existing models in the literature provide mutual information estimates, and by integrating these for insertion/deletion channels, mutual information can potentially be calculated, paving the way for refining and improving the existing bounds.

Another potential area for improvement is the application of newly developed deep learning models to construct an end-to-end communication scenario over insertion and deletion channels for both encoding and decoding. For example, transformers could be adopted for decoding problems, while generative adversarial networks or diffusion models could be applied to both encoding and decoding processes.

Finally, we note that reinforcement learning can be used to identify optimal encoding and decoding strategies. For example, it can help determine which marker codes are more effective in insertion and deletion channels, or it can detect particular deletion/insertion patterns and boost the decoding performance.

Bibliography

[1] E. U. Kargı and T. M. Duman, "A deep learning based decoder for concatenated coding over deletion channels," in *ICC 2024 - IEEE International Conference on Communications*, pp. 2797–2802, 2024.

[2] R. White, R. Newt, and R. Pease, "Patterned media: a viable route to 50 Gbit/in/sup 2/ and up for magnetic recording?," *IEEE Transactions on Magnetics*, vol. 33, no. 1, pp. 990–995, 1997.

[3] J. Hu, T. M. Duman, and M. F. Erden, "On the information rates of channels with insertion/deletion/substitution errors," in *2008 IEEE International Conference on Communications*, pp. 956–960, 2008.

[4] J. Hu, T. M. Duman, E. M. Kurtas, and M. F. Erden, "Bit-patterned media with written-in errors: Modeling, detection, and theoretical limits," *IEEE Transactions on Magnetics*, vol. 43, no. 8, pp. 3517–3524, 2007.

[5] H. J. Richter, A. Y. Dobin, R. T. Lynch, D. Weller, R. M. Brockie, O. Heinonen, K. Z. Gao, J. Xue, R. J. M. van der Veerdonk, P. Asselen, and M. F. Erden, "Recording potential of bit-patterned media," *Applied Physics Letters*, vol. 88, pp. 222512–1–222512–3, May 2006.

[6] S. M. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, vol. 1, no. 3, pp. 230–248, 2015.

[7] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[8] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. Leproust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[9] I. Shomorony and R. Heckel, "DNA-based storage: Models and fundamental limits," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3675–3689, 2021.

[10] M. Davey and D. Mackay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687–698, 2001.

[11] R. G. Gallager, "Sequential decoding for binary channels with noise and synchronization errors," tech. rep., MIT Lincoln Lab. Group, October 1961.

[12] J. Hu, T. M. Duman, M. F. Erden, and A. Kavcic, "Achievable information rates for channels with insertions, deletions, and intersymbol interference with i.i.d. inputs," *IEEE Transactions on Communications*, vol. 58, no. 4, pp. 1102–1111, 2010.

[13] L. Conde-Canencia and L. Dolecek, "Nanopore dna sequencing channel modeling," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 258–262, 2018.

[14] R. L. Dobrushin, "Shannon's theorems for channels with synchronization errors," *Problems of Information Transmission*, vol. 3, no. 4, pp. 11–26, 1967.

[15] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ: Wiley, 2nd ed., 2006.

[16] D. Fertonani and T. M. Duman, "Novel bounds on the capacity of the binary deletion channel," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2753–2765, 2010.

[17] D. Fertonani, T. M. Duman, and M. F. Erden, "Bounds on the capacity of channels with insertions, deletions and substitutions," *IEEE Transactions on Communications*, vol. 59, no. 1, pp. 2–6, 2011.

[18] M. Rahmati and T. M. Duman, "Bounds on the capacity of random insertion and deletion-additive noise channels," *IEEE Transactions on Information Theory*, vol. 59, no. 9, pp. 5534–5546, 2013.

[19] D. Fertonani and T. M. Duman, "Upper bounding the deletion channel capacity by auxiliary memoryless channels," in *2009 IEEE International Conference on Communications*, pp. 1–5, 2009.

[20] M. Rahmati and T. M. Duman, "Upper bounds on the capacity of deletion channels using channel fragmentation," *IEEE Transactions on Information Theory*, vol. 61, no. 1, pp. 146–156, 2015.

[21] E. Drinea and A. Kirsch, "Directly lower bounding the information capacity for channels with i.i.d. deletions and duplications," in *2007 IEEE International Symposium on Information Theory*, pp. 1731–1735, 2007.

[22] E. Drinea and M. Mitzenmacher, "Improved lower bounds for the capacity of i.i.d. deletion and duplication channels," *IEEE Transactions on Information Theory*, vol. 53, no. 8, pp. 2693–2714, 2007.

[23] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics. Doklady*, vol. 10, pp. 707–710, 1965.

[24] N. J. A. Sloane, "On single-deletion-correcting codes," *arXiv: Combinatorics*, 2002.

[25] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," *Automation and Remote Control*, vol. 26, no. 2, pp. 286–290, 1965. Translated to English.

[26] W. Ferreira, W. Clarke, A. Helberg, K. Abdel-Ghaffar, and A. Vinck, "Insertion/deletion correction with spectral nulls," *IEEE Transactions on Information Theory*, vol. 43, no. 2, pp. 722–732, 1997.

[27] M. Mansour and A. Tewfik, "Convolutional codes for channels with substitutions, insertions, and deletions," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 2, pp. 1051–1055 vol.2, 2002.

[28] T. Swart and H. Ferreira, "Insertion/deletion correcting coding schemes based on convolution coding," *Electronics Letters*, vol. 38, no. 16, p. 871, 2002.

[29] F. Sellers, "Bit loss and gain correction code," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 35–38, 1962.

[30] J. Chen, M. Mitzenmacher, C. Ng, and N. Varnica, "Concatenated codes for deletion channels," in *IEEE International Symposium on Information Theory, 2003. Proceedings.*, pp. 218–218, 2003.

[31] F. Wang, D. Fertonani, and T. M. Duman, "Symbol-level synchronization and LDPC code design for insertion/deletion channels," *IEEE Transactions on Communications*, vol. 59, no. 5, pp. 1287–1297, 2011.

[32] G. Ma, X. Jiao, J. Mu, H. Han, and Y. Yang, "Deep learning-based detection for marker codes over insertion and deletion channels," *IEEE Transactions on Communications*, pp. 1–1, 2024.

[33] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge: Cambridge University Press, 2009.

[34] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. USA: Prentice-Hall, Inc., 2004.

[35] G. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[36] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[37] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, 2017.

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

[39] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346, 2016.

[40] K. Cho, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997.

[42] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," *arXiv preprint arXiv:1805.09317*, 2018.

[43] Y. Jiang, S. Kannan, H. Kim, S. Oh, H. Asnani, and P. Viswanath, "Deepturbo: Deep turbo decoder," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, 2019.

[44] Y. Choukroun and L. Wolf, "Error correction code transformer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 38695–38705, 2022.

[45] Y. Liao, S. A. Hashemi, J. M. Cioffi, and A. Goldsmith, "Construction of polar codes with reinforcement learning," *IEEE Transactions on Communications*, vol. 70, no. 1, pp. 185–198, 2022.

[46] S. K. Mishra, D. Katyal, and S. A. Ganapathi, "A modified Q-learning algorithm for rate-profiling of polarization adjusted convolutional (PAC) codes," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2363–2368, 2022.

[47] E. Arikan, "From sequential decoding to channel polarization and back again," *CoRR*, vol. abs/1908.09594, 2019.

[48] L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "AI coding: Learning to construct error correction codes," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 26–39, 2020.

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

[50] S. Habib, A. Beemer, and J. Kliewer, "Belief propagation decoding of short graph-based channel codes via reinforcement learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 2, pp. 627–640, 2021.

[51] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[52] R. Pascanu, "On the difficulty of training recurrent neural networks," *arXiv preprint arXiv:1211.5063*, 2013.

[53] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[54] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?," *arXiv preprint arXiv:1908.01878*, 2019.

[55] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[56] D. J. MacKay, "Encyclopedia of sparse graph codes." Accessed: 2024-12-30.

[57] F. Wang, *Coding for Insertion/Deletion Channels*. Ph.D. Thesis, Arizona State University, Arizona, 2012.