UNIVERSITY OF CALIFORNIA SAN DIEGO

Machine Learning Techniques for Data Storage Systems: Modeling, Coding, and Detection

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering
(Communication Theory and Systems)

by

Simeng Zheng

2024

University of California San Diego

2024

DEDICATION

To my loves

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank many people for understanding, supporting, and helping me during my Ph.D. life at UCSD. It is challenging that half of my Ph.D. time was spent during the COVID pandemic.

First and foremost, I would like to sincerely thank my advisor, Professor Paul H. Siegel. As a lecturer, He is a professional advisor in coding theory series courses, leading us from course to research problems. he is the most responsible lecturer in ECE 250 and I am lucky to be the TA in his course. As an advisor, his insights on data storage and his guidance on research help me realize and overcome technical challenges in HDD and SSD. I am also grateful for all the on-campus, beach, thanksgiving, and virtual parties we have ever enjoyed. Paul is a devoted researcher, a brilliant mentor, a passionate lecturer, and a kind friend. I am deeply grateful to be his student.

I would appreciate many professors for their help over the years. I must thank Nuno Vasconcelos for the extraordinary machine learning series courses and for serving on my preliminary exam committee and defense committee. I would express my gratitude to Ramesh Rao for the valuable TA opportunity in ECE 250 and for serving on my preliminary exam committee and defense committee. I would appreciate Steven Swanson for serving on my defense committee and for organizing NVMW. I would like to thank Alon Orlitsky for the information theory course and for hosting beautiful ITA workshops. I am thankful for all lecturers: Robert Lugannani, Laurence B. Milstein, Young-Han Kim, Alexander Vardy, Xinyu Zhang, Manuela Vasconcelos, Todd Kemp, Niema Moshiri, Pengtao Xie, and Bryan Chin. I particularly enjoyed interesting discussions with Anxiao (Andrew) Jiang in ITA and NVMW. I am indebted to Ahmed Hareedy and Robert Calderbank for our wonderful collaboration on read-and-run constrained codes. I would thank Richard Wesel for organizing NASIT 2022 at UCLA and giving me an unforgettable information theory summer school. I would like to thank Brian M. Kurkoski for the stimulating discussion on ITA graduation day. I would thank Paul H. Siegel, Tara Javidi, Alon Oslitsky, and Ken Zeger for selecting me as the recipient of the 2023-2024 Shannon Graduate Fellowship. I sincerely appreciate Qin Huang at Beihang University for introducing me to the world of coding theory and giving me courage to explore further in this fantastic field.

It has been an exceptional experience for me to collaborate with those talented researchers: Yi Liu, Michael Baluja, Ahmed Hareedy, Robert Calderbank, Chih-Hui Ho, Wenyu Peng, Andrew Tan, Carolina Fernández, Ismael González Valenzuela. I am grateful to CMRR STAR Group members: Pengfei Huang, Yonglong Li, Yi Liu, Andreas Lenz, Wei Wu, Karthik Nagarjuna Tunuguntla, Navya Sree Prem, Shahar Dror, Yang Qiu, Michael Baluja, Ziwei Liu, Omer Sabary, Wenyu Peng, Sarah Ekaireb, Goldie Jin, Zachary Taylor Blair, Andrew Tan, and Ziyuan Zhu. We had so many exciting conversations in both research and life. I would appreciate help from the industry: Naoaki Kokubun, Hironori Uchikawa, Dongmin Shin, Sangho Yun. I would also thank my TAs: Bentao Zhang, Chih-Hui Ho, and Hanwen Yao for their high-level teaching professionalism.

Outside UCSD, I have been lucky to have internships in industry-leading companies: I would thank Joseph B. Soriaga, Arash Behboodi, and Jilei Hou at Qualcomm. Exploring generative models for wireless communication systems inspires several ideas in my thesis. I would also appreciate Ned Varnica, Erich Haratsch, and Mats Öberg at Marvell. My research particularly benefits from understanding NAND flash architecture during Marvell internship. The patents demonstrate the fruitful internships.

I would like to thank CMRR staff: Marina Robenko, Yvonne Gao, Octavio Ochoa, Raymond Descoteaux, and the cleaning staff, for providing me with a comfortable place for research.

I am super lucky to have my friends in San Diego: Jiaye Zhou, Chenlin Liu, Zhaoyuan He, Gehua Qin, Jiong Liu, Yang Qiu, Song Wang, Jingqi Huang, Kun Qian, Xin Xiao, Yang Fu, Chongbin Zhang, Susu Luo, Kan Wang, and Kaiyan Yan. My life at San Diego is colorful and joyful with your consistent company. I would also appreciate all my friends from Chongqing No. 1 Middle School and Beihang University. Our friendship which spans over ten years is an extremely precious treasure. I would thank all my friends for participating in my joys and sorrows throughout the years.

Most importantly, I would appreciate my parents for their love, support, and caring. I dedicate this dissertation to them.

The research was supported in part by the NSF under Grant CCF-2212437, Koxia Corporation, Marvell Technology, and the Center for Memory and Recording Research at the University of California San Diego.

Chapter 3 is in part a reprint of the material in the paper: Simeng Zheng, Chih-Hui Ho, Wenyu Peng, and Paul H. Siegel, "Flash-Gen: Spatio-temporal generator for flash memory systems," submitted to *IEEE Transactions on Communications*. The work was presented at DATE 2023: Simeng Zheng, Chih-Hui Ho, Wenyu Peng, and Paul H. Siegel, "Spatio-temporal modeling for flash memory channels using conditional generative nets," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, Antwerp, Belgium, Apr. 2023.

Chapter 4 is in part a reprint of the material in the paper: Simeng Zheng, Andrew Tan, Carolina Fernández, Ismael González Valenzuela, and Paul H. Siegel, "Optimal shaping codes for a TLC flash memory," in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024. The work was presented at Annual Non-Volatile Memories Workshop (NVMW) in Mar. 2024.

Chapter 5 is in part a reprint of the material in the paper: Ahmed Hareedy, Simeng Zheng, Paul H. Siegel, and Robert Calderbank, "Efficient constrained codes that enable page separation in modern flash memories," *IEEE Transactions on Communications*, vol. 71, no. 12, pp. 6834–6848, Dec. 2023. The work was also presented in part at ICC 2022 and NVMW 2024: A. Hareedy, S. Zheng, P. H. Siegel, and R. Calderbank, "Read-and-run constrained coding for modern flash devices," in *Proc. IEEE International Conference Communications (ICC)*, Seoul, South Korea, May 2022; Ahmed Hareedy, Simeng Zheng, Paul H. Siegel, and Robert Calderbank, "Read-and-run constrained coding for modern flash memories," in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024.

Chapter 6 is in part a reprint of the material in the paper: S. Zheng and P. H. Siegel,

"Code-aware storage channel modeling via machine learning," in *Proc. IEEE Information Theory Workshop (ITW)*, Mumbai, India, Nov. 2022, pp. 196–201. The work was presented at ITW 2022.

Chapter 7 is in part a reprint of the material in the paper: Simeng Zheng, Yi Liu, and Paul H. Siegel, "PR-NN: RNN-based detection for coded partial-response channels," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 1967–1982, July 2021. The work was also presented in part at ITW 2020: Simeng Zheng, Yi Liu, and Paul H. Siegel, "RNN-based detection for coded partial-response channels," in *Proc. IEEE Information Theory Workshop (ITW)*, Riva del Garda, Italy, Apr. 2021.

VITA

2014–2018	Bachelor of Engineering in Electronic and Information Engineering, Beihang University, China
2018–2020	Master of Science in Electrical and Computer Engineering (Communication Theory and Systems), University of California San Diego
2020–2024	Doctor of Philosophy in Electrical and Computer Engineering (Communication Theory and Systems), University of California San Diego
2023–2024	Shannon Graduate Fellowship

PUBLICATIONS

Simeng Zheng, Andrew Tan, Carolina Fernandez, Ismael González Valenzuela, and Paul H. Siegel, “Code-aware storage channel modeling via transfer learning,” to be submitted to *IEEE Transactions on Communications*.

Simeng Zheng, Chih-Hui Ho, Wenyu Peng, and Paul H. Siegel, “Flash-Gen: Spatio-temporal generator for flash memory systems,” submitted to *IEEE Transactions on Communications*.

Ahmed Hareedy, Simeng Zheng, Paul H. Siegel, and Robert Calderbank, “Efficient constrained codes that enable page separation in modern flash memories,” *IEEE Transactions on Communications*, vol. 71, no. 12, pp. 6834–6848, Dec. 2023.

Simeng Zheng, Yi Liu, and Paul H. Siegel, “PR-NN: RNN-based detection for coded partial-response channels,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 1967–1982, July 2021.

Simeng Zheng, Andrew Tan, Carolina Fernández, Ismael González Valenzuela, and Paul H. Siegel, “Optimal shaping codes for a TLC flash memory,” in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024.

Ahmed Hareedy, Simeng Zheng, Paul H. Siegel, and Robert Calderbank, “Read-and-run constrained coding for modern flash memories,” in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024.

Simeng Zheng, Chih-Hui Ho, Wenyu Peng, and Paul H. Siegel, “Spatio-temporal modeling for flash memory channels using conditional generative nets,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, Antwerp, Belgium, Apr. 2023.

S. Zheng and P. H. Siegel, “Code-aware storage channel modeling via machine learning,” in *Proc. IEEE Information Theory Workshop (ITW)*, Mumbai, India, Nov. 2022, pp. 196–201.

A. Hareedy, S. Zheng, P. H. Siegel, and R. Calderbank, “Read-and-run constrained coding for modern flash devices,” in *Proc. IEEE International Conference Communications (ICC)*, Seoul, South Korea, May 2022.

Simeng Zheng, Yi Liu, and Paul H. Siegel, “RNN-based detection for coded partial-response channels,” in *Proc. IEEE Information Theory Workshop (ITW)*, Riva del Garda, Italy, Apr. 2021.

Qin Huang, Simeng Zheng, Yuanhan Ni, Zulin Wang, and Shuai Wang, “Querying policies based on sparse matrices for noisy 20 questions,” in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Paris, France, July 2019.

ABSTRACT OF THE DISSERTATION

Machine Learning Techniques for Data Storage Systems: Modeling, Coding, and Detection

by

Simeng Zheng

Doctor of Philosophy in Electrical Engineering
(Communication Theory and Systems)

University of California San Diego, 2024

Professor Paul H. Siegel, Chair

As data explodes in modern applications, such as the Internet of Things (IoT), the needs for storage devices drastically increase. Solid state drives (SSDs) and hard disk drives (HDDs) are two main data storage devices. SSDs store data in flash memories, while HDDs store data in magnetic disks. To extend the lifetime and enhance the reliability of data storage devices, we utilize machine learning as the fundamental tool for three data storage modules: modeling in flash memory systems, error correction and constrained coding schemes, and detection method in magnetic recording channels.

The modeling part of the dissertation is devoted to proposing a novel data-driven approach, referred to as Flash-Gen, to generating NAND flash memory read voltages in both space and time using conditional generative networks. This generative modeling method reconstructs read voltages from an individual memory cell based on the program levels of the cell and its surrounding cells, as well as the time stamp, in a time-efficient, resource-saving, and function-comprehensive manner. As the needs for data-dependent channel models, we further extend the generative modeling approach to the coded storage channel. We train the generative models via transferring knowledge from models pre-trained with pseudo-random data. This technique can accelerate the training process and improve model accuracy in reconstructing the read voltages induced by constrained input data throughout the flash memory lifetime.

The coding part of the dissertation designs practical coding workflow and proposes new constrained and shaping coding schemes for flash memories. We propose a flash system optimization procedure, referred to as the Flash-Gen coding workflow, that leverages reconstructed read voltages from Flash-Gen for the development of error correction codes (ECCs) and constrained codes. Flash-Gen coding workflow can effectively address a range of important tasks, including threshold determination, coding performance estimation, and pattern characterization. We then formulate inter-cell interference (ICI)-mitigation constrained codes and distribution-matching shaping codes. The proposed coding schemes both achieve remarkable lifetime improvement.

The detection part of the dissertation builds recurrent neural network (RNN)-based detection for magnetic recording channels with partial-response equalization, which is referred to as *Partial-Response Neural Network* (PR-NN). PR-NN outperforms classical detection methods, such as the Viterbi detector, under multiple “realistic” environments and preserves the detection performance across different channel conditions.

Chapter 1

Introduction

1.1 Data Storage and Machine Learning

In recent years, machine learning has proven to be highly effective across various applications, particularly in the fields of computer vision and natural language processing. The ongoing revolution in artificial intelligence is profoundly reshaping the fields of data storage systems. We treat data storage devices, such as flash memories and magnetic drives, as a communication system. The communication system takes an input write signal and produces an output read signal. The coding techniques, including error correction codes (ECCs) and constrained codes, aim to recover the distorted data and mitigate various types of error sources in storage devices. Machine learning can be a critical tool to analyze and assist the necessary modules in data storage systems [51].

Understanding the NAND flash memory channel has become more and more challenging due to the continually increasing density and the complex distortions arising from the write and read mechanisms. Several mathematical models could reconstruct the read voltage distributions [11, 67, 78, 95] and hard bit errors [94, 112]. As system becomes more and more complicated, machine learning has been exploited to model flash memory channels [73, 75–77]. Machine learning-based modeling approaches are also proposed for wireless communication channels [5, 93] and DNA storage channels [53]. In this dissertation, we develop a spatio-temporal generative modeling approach for flash memory systems. Furthermore, we transfer the knowledge from the pretrained generative models to data-dependent cases, which accelerates the training process and preserves the reconstruction quality.

We then study a variety of error correction and mitigation mechanisms based on the understanding of channels. Error correction codes use parity check bits to detect and correct the bit errors that occur within flash memory. The typical ECC algorithm is low-density parity-check (LDPC) codes [10,46,123]. Constrained codes forbid error-prone patterns in both flash memories and hard disk drives to mitigate inter-cell and inter-symbol distortions distortions [1,23,30,34, 35,39,49,55,86,111,115,121,128]. Shaping codes, the distribution-matching codes, find several successful applications in flash devices by optimizing the wear costs associated with program levels [70–72, 104]. In this dissertation, the practical LDPC coding workflow is designed to optimize LDPC coding schemes for flash memory. Efficient constrained coding is formulated to enable page separation and the optimal shaping code is constructed to minimize the wear costs for the TLC flash device.

Exploring data storage systems and analyzing data protection schemes, we aim to study the data detection techniques for data storage systems. The classical detection method for magnetic recording channels with constrained coding is trellis-based sequence detection [4, 20, 21, 54, 117]. Deep neural networks are applied to communication and storage decoders [7, 26, 32, 52, 59, 68, 89, 101, 105, 106, 110]. The RNN-based detection, capable of streaming data inputs and preserving detection results under several situations, are proposed in this dissertation.

The synergy between machine learning and data storage systems is stimulating crucial mutual advancement [51]. Information theory is also important for machine learning models and their data. As models and data are stored in storage clusters, coding theory can protect model parameters and data from distortions [45].

1.2 Dissertation Overview

In this dissertation, we first introduce the data characterization platform for flash memory. Then, we propose generative modeling methods for flash memory systems. According to our error characterization for flash memories, we propose the optimal shaping coding scheme and the efficient constrained coding method for flash memories. We further design code-aware and data-dependent generative models. We finally establish the machine learning-based and robust detector for partial-response channels. The dissertation is organized as follows.

In Chapter 2, we introduce the FPGA platform for TLC flash memory characterization. We explore the flash conditions using hard read mode and soft read mode. We design the data-dependent workflow to incorporate coding schemes into the write information. The flash charaterization platform serves as the experimental foundation for all modeling, coding, and detection work in later chapters.

In Chapter 3, we propose Flash-Gen framework to generate flash memory read voltages in both space and time using conditional generative networks. We then utilize the reconstructed read voltages for the development of error correction codes (ECCs) and constrained codes. This coding development process can effectively address a range of important tasks, including threshold determination, coding performance estimation, and inter-cell interference (ICI) pattern characterization. The system modeling for flash memory motivates our researches on coding schemes in Chapter 4 and Chapter 5.

In Chapter 4, we establish the optimal shaping codes for a TLC flash memory. In TLC flash devices, three bits are stored in each cell to represent one of 8 program levels and wear costs for 8 levels are different during the repeated program and erase operations. The goal of shaping codes is to optimally shaping the probability distribution of sequences so that sequences with high wear costs have low probability and sequences with low wear costs have high probability. We demonstrate $10\times$ chip lifetime improvement based on Spanish-language texts.

In Chapter 5, we suggest new constrained coding schemes referred to as *read-and-run* (RR). The idea is to avoid error-prone patterns by coding data either only on the left-most page or only on the two left-most pages while leaving data on all remaining pages uncoded. The page separation idea preserves high access speed and the coding on one or two pages reduces the rate loss without impacting the device reliability. We verify RR codes using several constrained coding schemes, including lexicographically-ordered constrained (LOCO) codes, interleaved run-length limited (RLL) codes, and two-dimensional (2D) constrained codes. The notable performance gain is achieved via RR coding on our flash characterization platform.

In Chapter 6, we extend our Flash-Gen approach to the data-dependent storage channel. To reduce the experimental overhead associated with collecting extensive measurements from constrained program/read data, we train Flash-Gen models via transferring knowledge from models pre-trained with pseudo-random data. This technique can accelerate the training process and improve model accuracy in reconstructing the read voltages induced by constrained input data.

In Chapter 7, we investigate the use of recurrent neural network (RNN)-based detection of magnetic recording channels with inter-symbol interference (ISI). We refer to the proposed detection method, which is intended for recording channels with partial-response equalization, as *Partial-Response Neural Network* (PR-NN). We train bi-directional gated recurrent units (bi-GRUs) to recover the ISI channel inputs from noisy channel output sequences and evaluate the network performance when applied to continuous, streaming data. We demonstrate that the PR-NN detector outperforms Viterbi detection and achieves the performance of *Noise-Predictive Maximum Likelihood* (NPML) detection in additive colored noise (ACN) at different channel densities. We further show that PR-NN maintains its performance when jointly trained with two different channel conditions.

Chapter 2

Flash Characterization Platform

2.1 Measurement Method

As shown in Fig. 2.1, the flash characterization platform is an FPGA platform and can control and communicate with vendor triple-level cell (TLC) flash chips [8]. The developed software API can operate low-level hardware actions. We use the software API to simulate the write/read/erase operation cycle in flash chips.

The basic unit of data storage in NAND flash memory is a floating-gate transistor, referred to as a cell. The cells are organized into an interconnected 2-D array, called a block, via horizontal wordlines (WLs) and vertical bitlines (BLs). In our chip, each block has the right plane and the left plane. The flash memory chip is composed of a collection of such blocks. The basic unit of write and read operations in flash memory is a page, corresponding to a logical bit position in a wordline of a block. For TLC flash, the three pages are referred to as the lower, middle, and upper page. On the other hand, the basic unit of an erase operation is an entire block.

We briefly summarize functions:

- Program: write data on a wordline. Each wordline must be written three times as required by the TLC device's 3-stage program sequence.
- Erase: erase data in a block.
- Soft Read: read the voltage level for all cells on a page. The voltage level is fine-grained information about the voltage written to each cell.
- Hard Read: read the bit information for all cells on a page.

We conduct program/read/erase experiments as follows:

1. Erase flash memory block under test.
2. Program all pages of block under test with pseudo-random data.

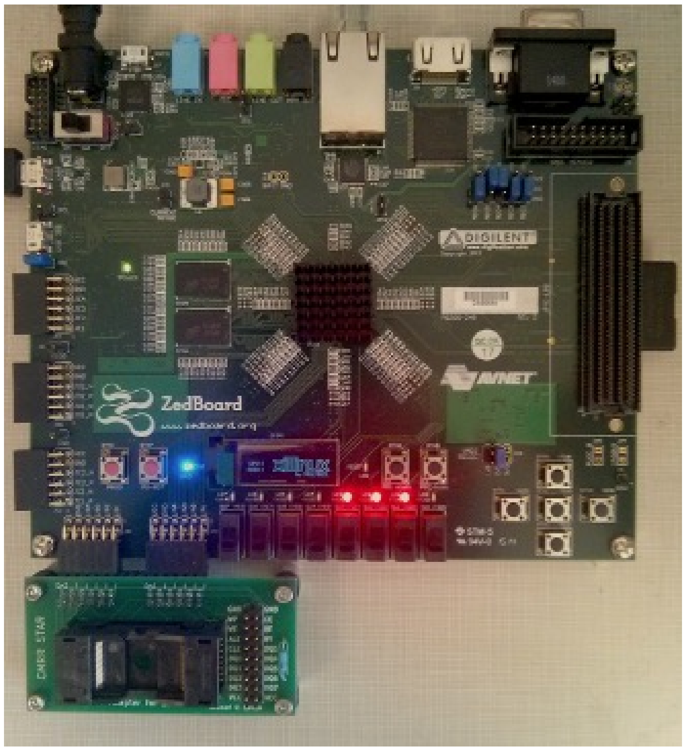

Figure 2.1. The FPGA testboard is the flash characterization platform.

3. We initially set values of read cycles. After the specific program/erase (P/E) cycles, perform a hard or soft read operation on the tested block. If we set retention time as 0, the read operation will be implemented instantly. Otherwise, the read operation will be implemented after the retention time.

4. For each cell, record the read information for post-processing.

The post-processing method for hard read signals focus on BER at different time stamps. In Chapter 5 and Chapter 4, we operate hard measurement experiments to verify bit error rate (BER) performance on designed coding schemes. The post-processing methods for soft read signals include probability density function (PDF) visualization, statistical analysis, error-prone patterns recognition, and read threshold determination. In Chapter 3 and Chapter 6, we operate soft measurement experiments to analyze measured and reconstructed signals.

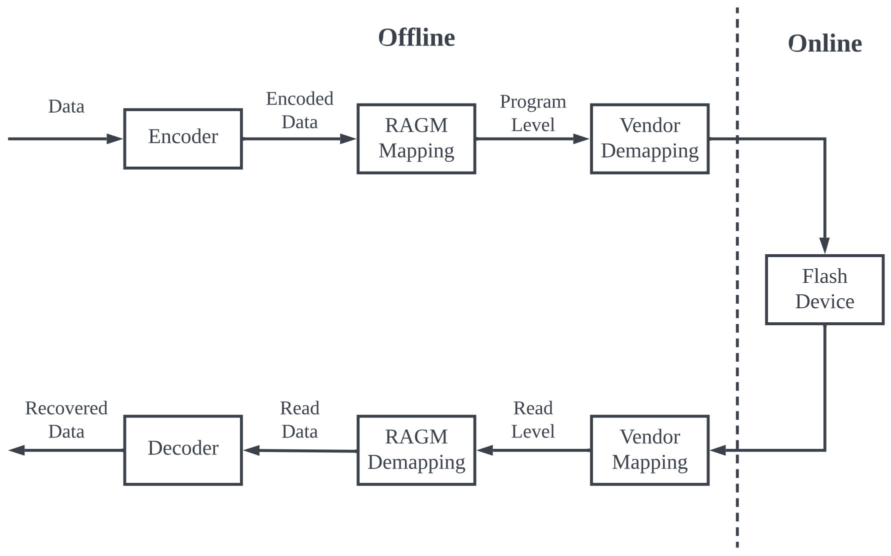

Figure 2.2. The system of coding incorporation module on flash characterization platform.

2.2 Coding Incorporation Module

We design the coding incorporation module to write encoded data instead of pseudo-random data into flash chip. The idea is to replace the memory of write variable with encoded data. We utilize Dump/Restore function in the Xilinx tool store.

We present the system of the coding incorporation module in Fig. 2.2. We perform program/read/erase experiments in the online part. We prepare coded data sequences and post-process read data in the offline part.

Due to the variation of mappings between manufacturers and product generations, we propose mapping conversion between the classical recursive alternate Gray mapping (RAGM) and vendor mapping to keep the consistency of programmed data. In TLC flash device, the mapping function converts 3 bits in one cell into 8 program levels and the demapping function represents 8 program levels into 3 information bits.

Chapter 3

Flash-Gen: Spatio-temporal Generator for Flash Memory Systems

3.1 Introduction

NAND flash memory is a non-volatile data storage medium, widely utilized in consumer electronics and large-scale data centers [10]. The basic storage unit in a flash memory is a floating-gate transistor, referred to as a cell. The cell can encode one or multiple bits of data as a discrete set of charge levels induced by a program operation. During a read operation, the program level is determined by measuring the threshold voltage of the cell. As the manufacturers scale down to a smaller flash memory chip and more bits are stored in one cell, identifying the voltage level stored in the cell becomes more and more difficult. This scaling of flash memory is accompanied by diminished memory reliability and reduced device endurance.

The distribution of read voltage levels corresponding to a program level is influenced by a variety of distortions. The distortions in NAND flash memory are manifold, including programming errors, inter-cell interference (ICI) due to parasitic capacitance coupling between adjacent cells, cell wear during program/erase (P/E) cycling, cell charge loss due to data retention, and program/read disturb effects. As the cell bit density increases and the gap between adjacent program levels decreases, these distortions and the reduced separation of the read voltage distributions of neighboring levels lead to an increase in error probability.

In order to compensate for the sophisticated noise and recover the stored bits in each cell, solid-state drives (SSDs), composed of a number of flash chips, incorporate advanced coding schemes within their controllers. Commonly used coding schemes in SSD controllers are error-correction codes (ECCs). Typical ECCs employed in flash controllers are Bose-Chaudhuri-Hocquenghem (BCH) codes and low-density parity-check (LDPC) codes [10, 46]. As the NAND flash devices wear out under repeatedly program and erase operations or perform read operations with extended wait time, the bit error rate (BER) of the flash chip increases. When BER exceeds the maximum number of errors that ECC can correct, it becomes impossible to fully recover the read information, resulting in data loss. The endurance of an SSD is determined by the number of program/erase (P/E) cycles that can be performed without uncorrectable errors.

Constrained codes are another type of coding scheme, which avoids patterns prone to errors caused by inter-cell interference (ICI) effects [34, 38, 39]. The primary cause of ICI stems from the charge propagation from cells programmed with high levels to neighboring cells programmed with lower levels. Constrained codes aim to characterize and forbid those problematic patterns in flash devices.

The characterization of complicated flash errors and the design of these reliability-enhancing coding algorithms rely upon a comprehensive understanding of the flash memory system. A common approach for system characterization is to treat a flash cell as a communication channel. The communication channel takes an input program level and produces an output voltage level. An accurate model of the channel is therefore an indispensable tool. Moreover, such a model can potentially obviate the need for hardware- and time-intensive data collection for use in evaluating and optimizing the coding algorithms.

Several mathematical and machine learning models supported by empirical measurements or simulated voltages in flash memory have appeared in the literature. Cai et al. [11] model the voltage distribution in 2-bit per cell MLC flash devices as a Gaussian distribution. Parnell et al. [95] proposed a parameterized Normal-Laplace mixture model that more accurately describes MLC flash read voltage distributions. Luo et al. [78] proposed another accurate and computationally more efficient model for MLC flash, based on a modified version of the Student's t-distribution and a temporal power law. Statistical analysis of hard bit errors in [94, 112] and characterization of dominant error patterns in [12, 67] offer additional empirical understanding of flash memories.

Recently, there has been great interest in the application of machine learning in communications and networking, including data storage. For example, machine learning can be used to design robust signal detectors in magnetic recording [128] and establish LDPC decoders with flexible code lengths and column weights [123]. Machine learning has also been exploited to model flash memory channels. Liu et al. [77] used a neural network (NN) to model simulated read voltages as a function of P/E cycles for one individual program level in isolated MLC flash cells. Liu et al. [75] provided flash memory conditions to a NN to model voltage distributions for 3D NAND flash. Liu et al. [76] used a time-dependent NN to predict page error counts in TLC flash. Liu et al. [73] generated errors in 3D NAND flash using a conditional generative adversarial network (GAN) architecture.

However, as effective as these mathematical or machine learning models have been in the scenarios to which they have been applied, none has provided an accurate model of both spatial and temporal characteristics of flash memory read voltages and none could serve as the basis for read channel design tasks such as read threshold optimization, error-prone pattern ranking, and performance evaluation of ECCs and constrained codes.

Generative modeling techniques such as the Generative Adversarial Network (GAN) [31], the Variational Auto-Encoder (VAE) [62], and diffusion models [43] have been successfully applied to several situations, such as image processing [50]. Generative modeling has now became an important tool for analyzing both wireless communication systems and DNA storage systems. For example, GAN has been applied to capture the channel impulse response distribution in multi-input and multi-output (MIMO) wireless communication channel [93] and VAE is an essential model to reconstruct authentic communication signals [5]. GAN has also been used to learn the error statistics of a DNA storage channel and to provide a simulation tool for developing improved error-correction coding and DNA sequence recovery algorithms [53].

In view of the demonstrated power of neural networks in learning complex multidimensional distributions, we propose the use of conditional generative nets as an approach to modeling flash memory read voltages in both space and time. The read voltages of each cell can be regenerated by the learned model from an array of program levels. We refer to this generative modeling method as Flash-Gen. We summarize our contributions in this work as follows:

1. We propose a comprehensive, flexible, and data-driven Flash-Gen model to accurately reconstruct soft read voltages at each individual cell unit of flash memory. This machine learning approach can be flexibly applied to flash memories of any technology generation

and chip feature size.

2. We formulate a temporally controllable, conditional VAE-GAN network to regenerate cell read voltages from the program levels of an array of flash memory cells and a time stamp representing the P/E cycle count and the retention time.

3. We propose the Flash-Gen coding workflow. This workflow simulates the “realistic” error correction environment with LDPC codes in an SSD controller and emulates multiple necessary steps in hard-decision and soft-decision decoding. It also estimates the relative frequencies of spatially-dependent and pattern-dependent ICI-induced hard read errors to help the design of constrained codes.

4. We validate the proposed Flash-Gen on a 1X-nm, 3-bit per cell (TLC) NAND flash chip across various wear conditions. The evaluation metrics used to compare the model outputs to measurements include the read voltage distributions at different time stamps, the measure of central dispersion of different program levels, and the error counts of different program levels.

5. We explore several essential applications of Flash-Gen coding workflow with practical LDPC codes and constrained codes. By evaluating LDPC decoding performance, analyzing the threshold selection during read process, and ranking error-prone patterns for constrained code design and evaluation, we demonstrate the consistency between Flash-Gen coding workflow and realistic NAND flash controller.

3.2 Flash Memory and Spatio-temporal Noise

3.2.1 Flash Memory Basics

We start from the basic unit cell in a flash chip. Today’s flash memories are capable of storing single or multiple bits (e.g., 2 to 5 bits) per cell, where the n-bit strings correspond to $q = 2^n$ program levels. The cells are organized into an interconnected two-dimensional (2D)

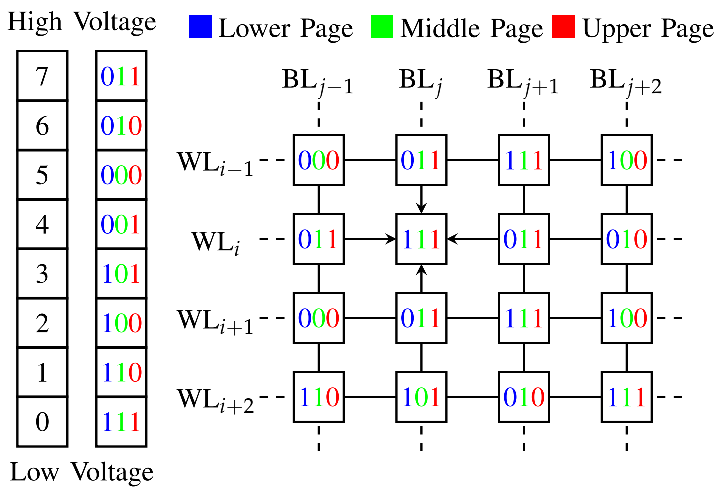

Figure 3.1. (Left) Example mapping of cell program levels to binary representations in TLC flash. (Right) Schematic diagram of a TLC flash memory block showing the 2-D array of cells connected in the horizontal direction by wordlines (WLs) and in the vertical direction by bitlines (BLs).

array, called a block, via horizontal wordlines (WLs) and vertical bitlines (BLs). The flash memory chip is composed of a collection of such blocks. In 3D NAND flash, these 2D arrays are stacked vertically to achieve larger volumetric density [74, 94]. Fig. 3.1 depicts a schematic diagram of a planar TLC flash memory block and an example of a Gray mapping from the $q = 8$ program levels to the corresponding data words of $n = 3$ binary digits.

The basic unit of write (i.e., program) and read operations in flash memory is a page, corresponding to a logical bit position in a wordline of a block. We refer to the program level as PL and the soft read voltage level as VL. On the other hand, the basic unit of an erase operation is an entire block.

In the program operation, we refer to the PLs of three consecutive cells along WLs or BLs as a pattern. As an example, in Fig. 3.1, the programmed levels $\text{PL}_{i-1,j}\text{PL}_{i,j}\text{PL}_{i+1,j}$ in WLs $(i-1), i, (i+1)$ of BL j, correspond to bit strings "011", "111", "011", which we associate with the pattern 707 in the vertical (BL) direction.

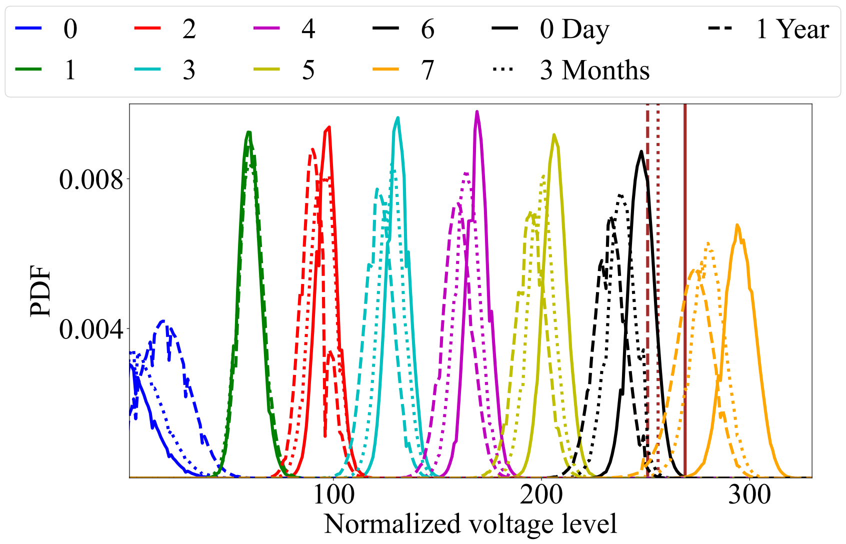

Figure 3.2. PDF visualizations of a 3D TLC NAND flash memory for measured voltage levels at no retention time, 3 months retention, and 1 year retention time when P/E cycle count is $10,000$ P/E. Solid curves, dotted curves, and dashed curves correspond to distributions of retention time 0 day, 3 months, and 1 year, respectively. Three brown vertical lines are the optimal read threshold between program level 6 (color black) and program level 7 (color orange).

3.2.2 Spatio-temporal Characteristics

Flash memory channels suffer from distortions of a spatio-temporal nature, where ICI effects correspond to spatial characteristics, P/E cycling noise and retention noise correspond to temporal characteristics.

Spatial effects: ICI refers to the phenomenon where programming of a cell induces changes in the voltage levels of neighboring cells within its block. In particular, the read voltage level of a cell programmed to a low level may be inadvertently increased if its adjacent cells are programmed to high levels, i.e., when the programming pattern is high-low-high. As an example in Fig. 3.1, if we program a pattern $\mathrm{PL}_{i,j-1}\mathrm{PL}_{i,j}\mathrm{PL}_{i,j+1} = 707$ in WL i or $\mathrm{PL}_{i-1,j}\mathrm{PL}_{i,j}\mathrm{PL}_{i+1,j} = 707$ in BL j in a TLC chip, the read voltage $\mathrm{VL}_{i,j}$ may be increased by its high adjacent cells. During data detection, the recovered program level of the central “victim” cell may therefore be erroneously interpreted as an incorrect level. ICI typically differs in the WL and BL directions.

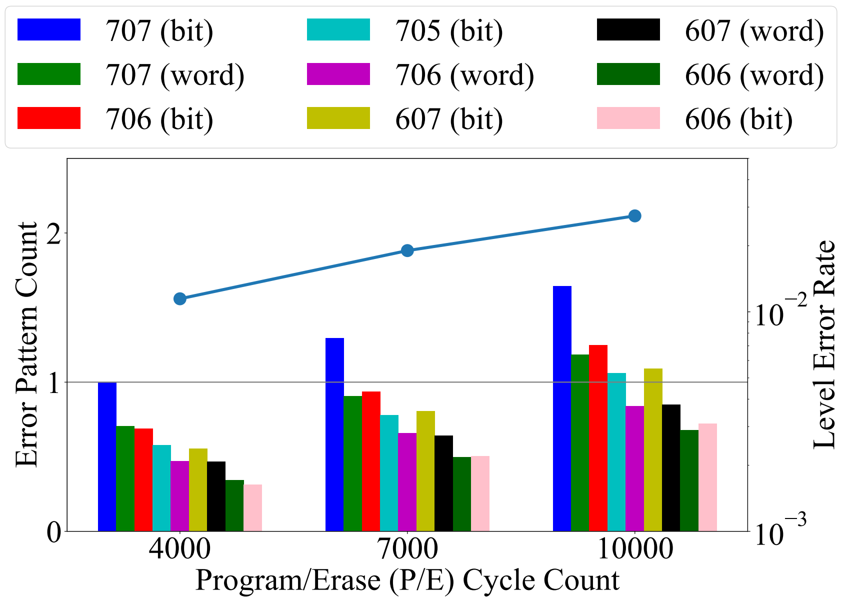

Figure 3.3. Count of top error-prone patterns and level error rate at selected P/E cycles without retention. The error pattern counts are normalized by the count of pattern 707 in BL direction at 4000 P/E cycles.

Temporal effects: P/E cycling gradually wears out the oxide layer of a cell. Data retention causes the programmed cell to diffuse electrons over time. As a result of these effects, the information stored in a cell can be misread. The changes of the read voltage distributions over the range of retention time are shown in Fig. 3.2. The retention noise systematically shifts the read thresholds to lower normalized voltage values as the retention time increases. For example, the read thresholds between program level 6 and program level 7 in three different retention timestamps are shown as brown vertical lines.

The integrated distortions of spatial ICI and temporal P/E cycling errors can be observed in Fig. 3.3. As the P/E cycle count increases, the error rate is increasing. For an individual P/E cycle count, the cell errors are clearly affected by neighboring program levels. Pattern 707 in the BL direction is the most severely affected by ICI. Moreover, patterns 707, 706, and 607 in the BL direction are more error-prone than those in the WL direction.

3.3 Flash-Gen Formulation

We define the flash memory system as a communication channel. Program level PL is from channel input domain $\mathcal{P}$ and voltage level VL is from channel output domain $\mathcal{V}$. The relationship between PL and VL can be represented as $\text{VL} = FM(\text{PL})$, where $FM(\cdot) : \mathcal{P} \rightarrow \mathcal{V}$ denotes the flash memory channel. More specifically, we consider the channel model in a rectangular region of a block,

$$\begin{bmatrix} & & \vdots & & \\ & \text{VL}_{i-1,j-1} & \text{VL}_{i-1,j} & \text{VL}_{i-1,j+1} & \\ \cdots & \text{VL}_{i,j-1} & \text{VL}_{i,j} & \text{VL}_{i,j+1} & \cdots \\ & \text{VL}_{i+1,j-1} & \text{VL}_{i+1,j} & \text{VL}_{i+1,j+1} & \\ & & \vdots & & \end{bmatrix} = FM\left(\begin{bmatrix} & & \vdots & & \\ & \text{PL}_{i-1,j-1} & \text{PL}_{i-1,j} & \text{PL}_{i-1,j+1} & \\ \cdots & \text{PL}_{i,j-1} & \text{PL}_{i,j} & \text{PL}_{i,j+1} & \cdots \\ & \text{PL}_{i+1,j-1} & \text{PL}_{i+1,j} & \text{PL}_{i+1,j+1} & \\ & & \vdots & & \end{bmatrix}\right),$$

where $\text{PL}_{i,j}$ and $\text{VL}_{i,j}$ correspond to the program level and voltage level of one cell, where i and j represent the WL and BL position of the cell, shown in Fig. 3.1.

When program level PL is given, voltage level $\text{VL}_{i,j}$ will be corrupted by both temporal and spatial factors. Temporal distortions arise from repeated program and erase operations, as well as from the time delay between program and read operations. Spatial distortions are caused by problematic program patterns giving rise to ICI.

In this section, we present the framework of our Flash-Gen model. As program levels from flash blocks and specific time stamps are provided, voltage levels can be sampled from Flash-Gen efficiently and accurately. We adopt a conditional VAE-GAN (cVAE-GAN) architecture [64] as the basis for our architecture. The network architecture is depicted in Fig. 3.4, where the fusion of the VAE [62] and GAN [31] can leverage the information from the latent space to produce high-quality and guarantee accurate reconstruction with the help of the discriminator. The spatio-temporal combination module incorporates temporal information, i.e., P/E cycles and retention into the voltage reconstruction. Our goal is to learn a mapping $FM(\cdot)$ between program levels and soft read voltage levels at various time stamps, where the reconstructed voltage levels accurately reflect the spatial and temporal nature of the channel.

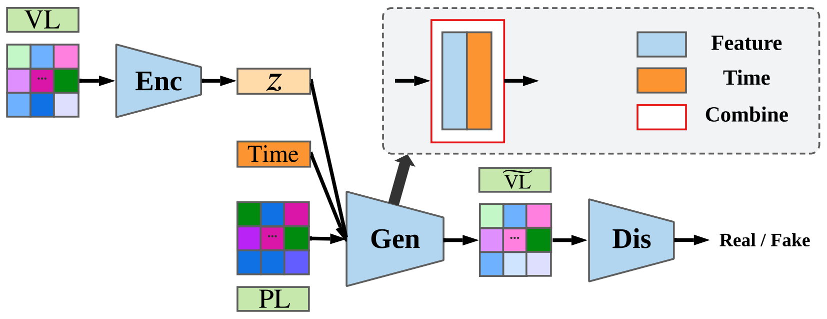

Figure 3.4. Flash-Gen architecture: encoder, generator, and discriminator constitute the generative modeling architecture. Here, z is the latent vector; Time is the corresponding time stamp (P/E cycle count and retention time discussed in this project); PL is the array of program levels, VL is the array of measured read voltage levels, and $\widetilde{\text{VL}}$ is the reconstructed array of read voltage levels.

3.3.1 Flash-Gen Architecture Formulation

Given program level PL $\in \mathcal{P}$, read voltage level VL $\in \mathcal{V}$, temporal factor time stamp representing P/E cycle count $T_{\text{P/E}}$ and the retention time T_γ, we aim to learn the analytically intractable likelihood $P(\text{VL}|\text{PL}, T_{\text{P/E}}, T_\gamma)$, with the goal of capturing the spatio-temporal nature of the flash memory channel.

Fig. 3.4 summarizes the architecture of our Flash-Gen. The conditional VAE-GAN architecture consists of three components: an encoder (Enc), a generator (Gen), and a discriminator (Dis). The encoder maps the read voltages to the latent vector z at a specific $T_{\text{P/E}}$ cycle count and retention time T_γ and replaces the prior distribution $P(z)$ in the GAN with the learned posterior distribution $P(z|\text{VL}, T_{\text{P/E}}, T_\gamma)$. The decoder in the VAE shares its weights with the GAN generator [31]. In the conditional setting, the variational lower bound of $P(\text{VL}|T_{\text{P/E}}, T_\gamma)$ can be derived as

$$\begin{aligned}\log P(\text{VL}|T_{\text{P/E}}, T_\gamma) \geq &- D_{KL}(Q(z|\text{VL}, T_{\text{P/E}}, T_\gamma)||P(z|T_{\text{P/E}}, T_\gamma)) \\ &+ \mathbb{E}_{Q(z|\text{VL}, T_{\text{P/E}}, T_\gamma)}[\log(P(\text{VL}|z, T_{\text{P/E}}, T_\gamma))],\end{aligned} \tag{3.1}$$

where D_{KL} represents the Kullback-Leibler (KL) divergence. The distribution $Q(z|\text{VL}, T_{\text{P/E}}, T_\gamma)$ of the latent vector z is trained to approach $P(z|T_{\text{P/E}}, T_\gamma)$ via the KL loss $\mathcal{L}_{KL}$, where

$P(z|T_{\text{P/E}}, T_\gamma)$ is assumed to be a multivariate Gaussian distribution.

Generator will take both the learned latent vector and PL as input and generate a "fake" $\widetilde{\text{VL}}$. The latent vectors are sampled from $Q(z|\text{VL}, T_{\text{P/E}}, T_\gamma)$ using the re-parameterization trick [62]. When sampling different latent vectors z from the same distribution, we can generate multiple arrays of plausible voltages levels. The variations in these output arrays for a given array of program levels reflect the stochasticity of the channel. The discriminator measures the difference between PL and $\widetilde{\text{VL}}$. The loss in the conditional GAN part is

$$\mathcal{L}_{GAN} = \log(1 - Dis(\text{PL}, Gen(\text{PL}, T_{\text{P/E}}, T_\gamma, z))) + \log(Dis(\text{PL}, \text{VL})). \tag{3.2}$$

Similar to VAE-GAN [64], we encourage the reconstructed voltage levels to match the authentic voltage levels, using the ℓ_2-norm to measure the reconstruction loss

$$\mathcal{L}_{recon} = ||\text{VL} - Gen(\text{PL}, T_{\text{P/E}}, T_\gamma, z)||_2. \tag{3.3}$$

Combining these equations, we formulate the loss function of the cVAE-GAN architecture as

$$\min_{Gen,Enc} \max_{Dis} \mathcal{L}_{GAN} + \alpha\mathcal{L}_{recon} + \beta\mathcal{L}_{KL}. \tag{3.4}$$

3.3.2 Spatio-temporal Fusion

To capture the spatial ICI effects in the channel model, we implement the generator using a convolutional neural network (CNN), where VL is reconstructed from the PL values in its cell and neighboring cells. To generate VL at an explicit time stamp, we control the generator with an additional temporal factor and incorporate the time stamp into generator Gen.

In order for generator to be aware of time stamps, we inject some encoded information about the time inputs. There are many choices of value expressions. We represent the time inputs as a d-dimensional vector using powers or exponential values. We encode the normalized

P/E cycle count using expressive powers, e.g., $T_{\text{P/E}}^2$, $\sqrt{T_{\text{P/E}}}$, etc. The d-dimensional vector of P/E cycle can be formulated as $[T_{\text{P/E}}^{0.5}, T_{\text{P/E}}^{1}, T_{\text{P/E}}^{1.5}, ...]$. Inspired by the modeling of retention noise in [67], we represent the normalized retention time using various exponential values $e^{-\nu T_\gamma}$, where ν is a constant parameter. The encoded vector can be presented as $[e^{-1T_\gamma}, e^{-0.75T_\gamma}, e^{-0.5T_\gamma}, ...]$.

We illustrate the fusion mechanism in generator using controllable P/E cycle count. We first encode the normalized P/E cycle count into a d-dimensional P/E vector, which contains expressive powers of the normalized P/E cycle. Then, we spatially replicate the d-dimensional P/E vector to the feature map with appropriate size $H \times W \times d$ and concatenate it with the $H \times W \times C$ feature from each layer in Gen, where $H \times W$ is the spatial dimension of the feature from each convolutional layer and C is the number of channels in the CNN. The channel-wise fusion produces the final feature with size $H \times W \times (C + d)$ of each layer. The fusion of the features from the program levels and the P/E feature maps guarantees the spatial-temporal characteristics of the reconstructed voltage levels. The controllable retention time shares the same fusion architecture with different encoded time vectors.

3.3.3 Implementation Details

Datasets: We validate the proposed Flash-Gen using datasets collected from one commercial 1X-nm TLC flash chip. We prepare two datasets for verification: P/E cycling dataset and retention dataset. We conduct the P/E cycling experiment by erasing a block, programming pseudo-random data, and reading voltage levels at selected P/E cycle counts with fixed retention time. We record the $\{\text{PL}, \text{VL}\}$ dataset at selected P/E cycles 4000, 7000, and 10000 with immediate reads. For the retention datasets, we repeatedly erase and program blocks with pseudo-random data to predefined P/E cycle counts, and read voltage levels after a certain retention time at room temperature. We record the $\{\text{PL}, \text{VL}\}$ dataset at 4000 P/E cycles with 0 retention time, τ retention time, and 2τ retention time.

In order to fit blocks of recorded data into the network architecture, we crop the $\{\text{PL}, \text{VL}\}$ pairs of the recorded blocks into non-overlapping 64×64 2-D arrays. In P/E cycling dataset, the number of 2-D arrays in the training set is 1.5×10^5 (5×10^4 for each P/E cycle) and the size of the evaluation dataset is 2.1×10^4 (7×10^3 for each P/E cycle). In retention dataset, the number of 2-D arrays in the training set is 9×10^4 (3×10^4 for each retention time) and the size of the evaluation dataset is 2.1×10^4 (7×10^3 for each retention time).

Network: The three network modules in Fig. 3.4 refer to: ResNet [41] (Enc), U-net [99] (Gen), and PatchGAN [50] (Dis). The dimensions of latent vector z and time vector are both set to 6. The following descriptions of the modules exploit the terminologies in the corresponding references.

1. Encoder: We use the two residual blocks, each of which contains two 3×3 convolutional layers with stride 1 and padding 1. We then add two linear layers, which map output features to mean and variance for the latent vector.

2. Generator: Ck denotes a Convolution-BatchNorm-ReLU layer with k output channels. All convolutions are 4×4 kernels applied with stride 2 and padding 1. The network architecture before spatio-temporal fusion can be described as

$$\text{(Down Part) } C64, C128, C256, C512, C512, C512,$$

$$\text{(Up Part) } C512, C512, C256, C128, C64, C1.$$

   where we inject latent vector z by replication and concatenation into every layer in the "Down" part [132], and each layer in the "Up" part receives skip connections from the corresponding layer in the "Down" part [99].

3. Discriminator Dis: The input to the discriminator is the concatenation of fake voltage levels and program levels. With the same naming convention as in the generator, we express the discriminator as $C64, C128, C1$.

We compared the cVAE-GAN model to other popular generative modeling architectures: conditional GAN [50], conditional VAE [108], and Bicycle GAN [132]. When we compare the total variational distance of voltage distributions between measured data and sampled data from the evaluation dataset, the cVAE-GAN model reconstructs the read voltage levels with the highest quality. The numerical results of comparison are not included due to the space limitation.

Remark 1. *Flash-Gen can be generalized to any multi-level flash device including 3D NAND flash. In order to characterize the severe layer-to-layer interference in 3D flash [73, 94], we can crop the 3D arrays of collected $\{PL, VL\}$ pairs and modify the Flash-Gen network realization to include 3D convolutional networks in all of the modules.*

Learning: We trained Flash-Gen models given dataset from scratch. We settled upon the training parameters after several experiments. Adam optimizer is used with learning rate 2×10^{-4}. Parameters in the loss function (3.4) are set to $\alpha = 10$ and $\beta = 0.01$. The P/E cycling Flash-Gen is trained with recorded $\{\mathrm{PL}, \mathrm{VL}\}$ dataset at P/E cycle counts 4000, 7000, and 10000. The total number of iterations is 5.25×10^5 for P/E cycling Flash-Gen and the batch size is 2. The retention Flash-Gen is trained with a dataset reflecting retention times 0, τ, and 2τ, where τ is chosen to observe noticeable differences in level distributions at the three time stamps. The total number of iterations number is 6.3×10^5 and the batch size is again 2.

During evaluation, we use program levels and latent vector z sampled from a standard multivariate Gaussian distribution. For each program level array, we sample 10 different latent vectors to evaluate the learned model.

Time: A major benefit of Flash-Gen is observed by comparing the measurement time from the real chip and the inference time of Flash-Gen. When we collect one block of data at selected P/E cycles with no retention age, it requires approximately 4 hours on our FPGA-based platform and the block cannot store information for further usage due to the high error rates. The retention datasets require waiting time to record voltage levels. However, in the Flash-Gen, the data generation of one block can be completed within 400 seconds under CPU mode (Intel i7-9700K, 3.60GHz$\times$8) and no flash block is wasted.

3.4 Flash-Gen Coding Workflow

Preserving data integrity in NAND flash involves dealing with temporal and spatial distortions. Coding is a powerful technique for mitigating, detecting, and correcting errors in data storage. Designing appropriate coding schemes takes several factors into consideration, like error correction capability, storage overhead, and read latency. To search for optimal coding design, a large number of experiments regarding threshold determination and code optimization are needed, which consumes excessive time and hardware resources. Read voltages sampled from classical mathematical distributions or generated by machine learning-based models do not accurately reflect the spatio-temporal characteristics of flash memory channels.

Therefore, we propose the Flash-Gen coding procedure. Flash-Gen coding workflow simulates three practical coding applications: hard-decision decoding of LDPC codes, soft-decision decoding of LDPC codes, and optimization of constrained codes. The detailed task modules are illustrated in Fig. 3.5. Our applications are not limited to these three coding tasks and can be extended to other signal-processing scenarios. For example, reconstructed $\{\text{PL, VL}\}$ containing spatio-temporal information can help the design of maximum a posteriori (MAP) and Gaussian approximation (GA) detectors [2].

3.4.1 Flash-Gen LDPC Coding Workflow

We first discuss the use of Flash-Gen in the evaluation of ECCs, specifically, LDPC codes. Modern flash devices employ advanced ECCs within their controllers to detect and correct a number of errors when data is read out from flash memory. A stronger ECC can tolerate more errors but consumes more power and latency.

We briefly overview the two stages of error correction and their related data recovery tasks [10]: 1) Hard-decision decoding: The flash chip reads each page using the set of hard thresholds, where the flash chip compares the read voltage with thresholds and determines the read bit for each page. Then ECC decoder performs the hard-decision decoding process on the

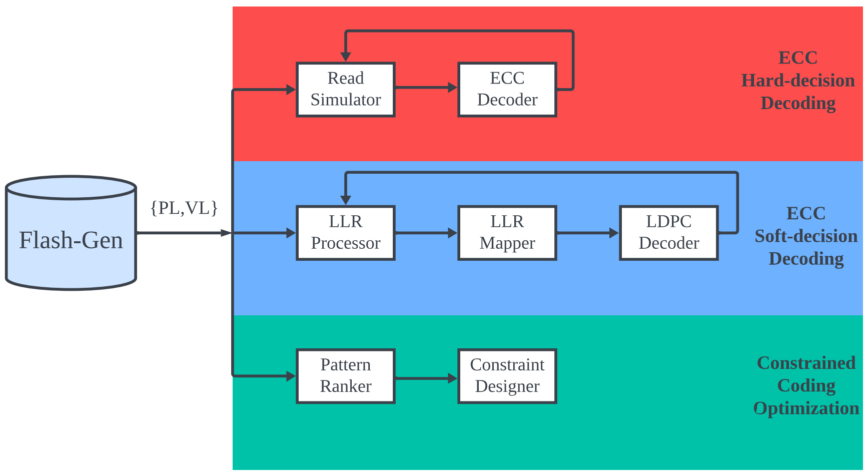

Figure 3.5. Flash-Gen coding workflow: Flash-Gen serves as a database to provide unlimited accurate $\{PL, VL\}$ pairs for coding techniques. We propose error correction coding flow with both hard-decision decoding mode and soft-decision decoding mode and constrained coding flow.

read bits. If the decoder succeeds in correcting all errors, the controller transmits the data to the host. Otherwise, the SSD controller steps into the more powerful second stage, which is soft-decision decoding. The main tasks in hard-decision decoding include threshold determination and decoding performance estimation; 2) Soft-decision decoding: The chip employs multiple thresholds between adjacent program levels during the read process, yielding soft information for each cell. The soft information, referred to as a log-likelihood ratio (LLR), estimates the posterior probability of a specified page bit being 0 or 1, given the read voltage. Since more read threshold comparisons lead to more fine-grained read information, the strength of soft-decision decoding relies on the number of read thresholds. The main tasks consist of LLR value computation, soft-decision threshold determination, and decoding performance estimation.

These ECC-related tasks rely on prior information regarding the flash memory channel and research on ECCs requires a huge amount of realistic readback signals. The Flash-Gen model is a natural candidate to assist and supplement the coding optimization functions, including estimating the complicated channel information and reducing heavy measurement workloads.

We now describe the Flash-Gen LDPC coding workflow to simulate the LDPC decoding procedure in flash devices, as represented schematically in Fig. 3.5. We formulate our approach using a TLC flash chip with $q = 8$ program levels. The learned Flash-Gen can generate unlimited pairs of $\{\text{PL}, \text{VL}\}$. We concatenate those 64×64 2-D arrays and reconstruct the flash blocks. The probabilities at the cell level can be expressed as

$$P(\text{PL} = pl, \text{VL} = vl) = \frac{N(pl, vl)}{\sum_{pl \in \mathcal{P}} \sum_{vl \in \mathcal{V}} N(pl, vl)}, \tag{3.5}$$

$$P(\text{PL} = pl) = \frac{\sum_{vl \in \mathcal{V}} N(pl, vl)}{\sum_{pl \in \mathcal{P}} \sum_{vl \in \mathcal{V}} N(pl, vl)}, \tag{3.6}$$

$$P(\text{VL} = vl | \text{PL} = pl) = \frac{N(pl, vl)}{\sum_{vl \in \mathcal{V}} N(pl, vl)}, \tag{3.7}$$

where $N(pl, vl)$ counts the number of cells with the program level pl and voltage level vl and $\sum_{pl \in \mathcal{P}} \sum_{vl \in \mathcal{V}} N(pl, vl)$ represents the total number of reconstructed cells. Conditional probability $P(\text{VL}|\text{PL})$ is calculated from the first two probabilities.

In the hard-decision decoding stage, we design the read simulator to perform read threshold optimization and process operation for each cell. The hard read bits are determined by comparing the voltage level VL to the thresholds. The set of hard read thresholds is presented as $\text{TH}_H = \{\text{TH}_{H,k} : k = 0, 1, ..., q-2\} \subseteq \mathcal{V}$, where $\text{TH}_{H,k}$ is the threshold to distinguish read bit between program level k and $k+1$. For instance, if a voltage level of PL 7 in Fig. 3.2 is lower than the last threshold $\text{TH}_{H,6}$ between level 6 and level 7, the hard read level of the cell will not be designated as 7. Indeed, there is an error if a voltage level of 1 lies below $\text{TH}_{H,0}$ or above $\text{TH}_{H,1}$. We compute the level error probability LER of $\text{PL} = 7$ and $\text{PL} = 1$ as

$$\begin{aligned}
\text{LER}(\text{TH}_H, \text{PL} = 7) &= P(\text{VL} < \text{TH}_{H,6} | \text{PL} = 7) = \frac{\sum_{vl < \text{TH}_{H,6}} N(\text{PL} = 7, vl)}{\sum_{vl \in \mathcal{V}} N(\text{PL} = 7, vl)}, \\
\text{LER}(\text{TH}_H, \text{PL} = 1) &= P(\text{VL} < \text{TH}_{H,0} | \text{PL} = 1) + P(\text{VL} > \text{TH}_{H,1} | \text{PL} = 1) \\
&= \frac{\sum_{vl < \text{TH}_{H,0}} N(\text{PL} = 1, vl)}{\sum_{vl \in \mathcal{V}} N(\text{PL} = 1, vl)} + \frac{\sum_{vl > \text{TH}_{H,1}} N(\text{PL} = 1, vl)}{\sum_{vl \in \mathcal{V}} N(\text{PL} = 1, vl)}.
\end{aligned} \tag{3.8}$$

We can compute the total level error rate as $\text{LER}(\text{TH}_H) = \sum_{pl \in \mathcal{P}} \text{LER}(\text{TH}_H, \text{PL} = pl)$. The bit error rate $\text{BER}(\text{TH}_H)$ will be converted from level errors by the label mapping in Fig. 3.1. To find the optimal read threshold for hard-decision decoding, we minimize the bit error rate of the read operation

$$\text{TH}_H^* = \arg\min_{\text{TH}_H} \text{BER}(\text{TH}_H). \tag{3.9}$$

Using the optimal read threshold, data are read from each cell and controller performs hard-decision decoding on the noisy data.

In the soft-decision decoding phase, we formulate the modules of LLR processor and LLR mapper. LLR processor determines the number of thresholds between two adjacent program levels and optimizes the threshold positions. The set of soft read thresholds can be denoted as $\text{TH}_S = \{\text{TH}_{S,k} : k = 0, 1, ..., (q-1)*l-1\} \subseteq \mathcal{V}$, where l is the number of reads between two program levels. If the number of reads is 1, the soft read voltage VL_S is equal to the hard read voltage VL_H. The additional number of reads l adds more accurate information about the program level and enhances the power of the LDPC decoder. We present the transition probability between two soft thresholds as

$$P(\text{TH}_{S,k} \leq \text{VL} < \text{TH}_{S,k+1} | \text{PL} = pl) = \frac{\sum_{\text{TH}_{S,k} \leq vl < \text{TH}_{S,k+1}} N(\text{PL} = pl, vl)}{\sum_{vl \in \mathcal{V}} N(\text{PL} = pl, vl)}. \tag{3.10}$$

We find the optimal placement of the soft threshold by maximizing the mutual information of the flash memory channel [96, 118]. The mutual information between program level PL and voltage level VL is given by

$$\begin{aligned} \text{MI}(\text{TH}_S) = I(\text{PL}, \text{VL}) &= H(\text{VL}) - H(\text{VL}|\text{PL}) \\ &= H(\sum_{k=0}^{(q-1)*l} p_{0,k}, \sum_{k=0}^{(q-1)*l} p_{1,k}, ..., \sum_{k=0}^{(q-1)*l} p_{q-1,k}) \\ &- \sum_{pl=0}^{q-1} p_{pl} H(\frac{p_{pl,0}}{p_{pl}}, \frac{p_{pl,1}}{p_{pl}}, ..., \frac{p_{pl,(q-1)*l}}{p_{pl}}), \end{aligned} \tag{3.11}$$

where H is the entropy function. the transition probability $p_{pl,k}$ represents $P(\text{TH}_{S,k} \leq \text{VL} < \text{TH}_{S,k+1}|\text{PL} = pl)$ and $p_{pl,(q-1)*l}$ is $P(\text{TH}_{S,(q-1)*l-1} \leq \text{VL}|\text{PL} = pl)$. The probability of each level p_{pl} is expressed as $P(\text{PL} = pl)$.

The optimization of soft thresholds TH_S can be formulated as

$$\text{TH}_S^* = \arg\max_{\text{TH}_S} \text{MI}(\text{TH}_S). \tag{3.12}$$

After determining the positions of soft thresholds, we need to compute the LLR values for each cell based on its voltage region. SSD controllers commonly establish an LLR table by utilizing either previously collected data or mathematical modeling distributions like the Gaussian model. However, LLR tables might not provide a precise and timely representation of the channel information. In the LLR mapper module, the Flash-Gen model can be used to compute the LLR values at runtime,

$$LLR(vl) = \log \frac{P(B(PL) = 0|\text{VL})}{P(B(PL) = 1|\text{VL})} = \log \frac{\sum_{B(pl)=0} N(\text{PL} = pl, vl)}{\sum_{B(pl)=1} N(\text{PL} = pl, vl)}, \tag{3.13}$$

where $B(.)$ maps the program level to the information bit on the corresponding page.

According to our designed soft-decision decoding procedures, we process streaming $\{\text{PL}, \text{VL}\}$ pairs to generate soft LLRs. In the LDPC decoding module, we assume the LDPC codeword is all 0. We then rewrite (3.13) as

$$LLR(vl) = \log \frac{P(0 + B(PL)|\text{VL})}{P(1 + B(PL)|\text{VL})}. \tag{3.14}$$

If the bit value for a page is 1, denoted as $B(PL) = 1$, then the LLR for the cell is $-LLR(vl)$. If $B(PL) = 0$, the LLR for the cell is $LLR(vl)$. The computed LLRs for each cell are computed and sent to the LDPC decoder as the flash memory channel output. If the output of the LDPC decoder is all 0 within a specified number of decoding iterations, we declare the decoding process successful; otherwise, we declare a decoding failure. We evaluate the LDPC coding workflow using both measured datasets and generated datasets in Section 3.6.1, where we use both measured and reconstructed sequences for read threshold determination, hard and soft information calculation, and LDPC decoder simulation.

We summarize the advantages of the Flash-Gen LDPC coding workflow as follows: 1) Flash-Gen greatly saves time and hardware resources in ECC performance evaluation and optimization. An essentially unlimited supply of $\{\mathrm{PL}, \mathrm{VL}\}$ pairs provides the opportunity to optimize code designs, read strategies, and decoding algorithms without any on-device operations. 2) "Realistic" $\{\mathrm{PL}, \mathrm{VL}\}$ pairs for each cell provide more authentic simulations for coding schemes than read voltages generated using mathematical models or previously proposed machine-learning models.

3.4.2 Flash-Gen Constrained Coding Workflow

ECC schemes have been employed to ensure the reliability of flash memory. However, spatio-temporal distortions of flash memory channels possess an asymmetric nature. ECC schemes assuming an underlying symmetric noise distribution may not be the most efficient approach. According to the spatio-temporal understanding of the dominant errors by Flash-Gen, it is crucial to design targeted coding schemes to combat dominant errors in flash memory. Constrained coding, by forbidding severe error-prone patterns, is an ideal candidate to mitigate the ICI effects. The key step of designing constrained codes is to determine the set of patterns that need to be forbidden. However, as the wear condition changes in the flash device and the binary representation of program levels is vendor-dependent, it is hard to design a universal constrained code suitable for the entire lifetime of a flash device.

We propose the Flash-Gen constrained coding workflow to address this design problem. We first apply the pattern ranker module, which arranges the pattern-dependent probabilities in descending order. Then, the constraint designer module will select the most error-prone patterns to formulate coding schemes. Based on the two-dimensional structure in Fig. 3.1, we examine the errors associated with program level patterns $\mathrm{PL}_{i,j-1}\mathrm{PL}_{i,j}\mathrm{PL}_{i,j+1}$ in the WL direction and

$\text{PL}_{i-1,j}\text{PL}_{i,j}\text{PL}_{i+1,j}$ in the BL direction. As we observed from our experiments, severe ICI errors follow the high-low-high program level pattern. We consider the problematic patterns with a central victim cell as 0 and calculate the pattern-dependent error probabilities in WL and BL directions as

$$\begin{aligned}
&P(\text{PL}_{i,j-1}, \text{PL}_{i,j} = 0, \text{PL}_{i,j+1} | \text{VL}_{i,j} > \text{TH}_{H,0}, \text{PL}_{i,j} = 0) \\
&= \frac{\sum_{vl > \text{TH}_{H,0}} N(\text{PL}_{i,j} = 0, vl | \text{PL}_{i,j-1}, \text{PL}_{i,j+1})}{\sum_{vl > \text{TH}_{H,0}} N(\text{PL}_{i,j} = 0, vl)}, \\
&P(\text{PL}_{i-1,j}, \text{PL}_{i,j} = 0, \text{PL}_{i+1,j} | \text{VL}_{i,j} > \text{TH}_{H,0}, \text{PL}_{i,j} = 0) \\
&= \frac{\sum_{vl > \text{TH}_{H,0}} N(\text{PL}_{i,j} = 0, vl | \text{PL}_{i-1,j}, \text{PL}_{i+1,j})}{\sum_{vl > \text{TH}_{H,0}} N(\text{PL}_{i,j} = 0, vl)},
\end{aligned} \tag{3.15}$$

where $N(\text{PL}_{i,j} = 0, vl | \text{PL}_{i,j-1}, \text{PL}_{i,j+1})$ and $N(\text{PL}_{i,j} = 0, vl | \text{PL}_{i-1,j}, \text{PL}_{i+1,j})$ count the number of cells with program level 0 and voltage level vl given the condition on the program levels on its neighboring cells.

We illustrate the pattern-dependent probabilities using our TLC flash device. We have 64 combinations of neighboring program levels of a given cell in both the WL and BL directions. Once we rank the pattern-dependent probabilities, we can summarize the most severe patterns and propose targeted constrained coding scheme for flash device. Examples illustrating the determination of problematic patterns and the design of constrained codes can be found in [35,38].

3.5 Statistical Analysis of Flash-Gen

During the inference process using the evaluation dataset, we denote recorded pairs $\{\text{PL}, \text{VL}\}$ from the flash device as our measured data and denote the reconstructed pairs $\{\text{PL}, \widetilde{\text{VL}}\}$ from Flash-Gen as the generated data. To evaluate the quality of the reconstructed voltage levels and analyze the spatio-temporal nature of flash memory system, we discuss the regenerated VL using the following statistical metrics:

1. Distribution: The frequency of occurrence of each voltage level given the program level,

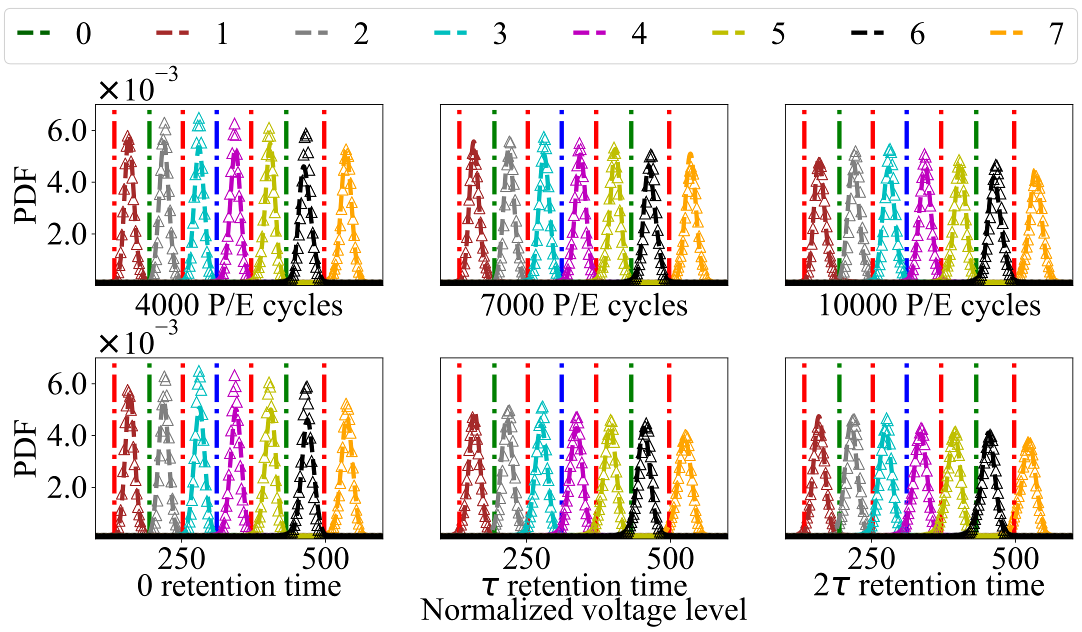

Figure 3.6. PDF visualizations for measured and Flash-Gen generated voltage levels at various time stamps: 4000, 7000, and 10000 P/E cycles; 0, τ, and 2τ retention time. The first row of subfigures corresponds to the P/E cycling dataset and the second row corresponds to the retention dataset. In each subfigure, dashed curves represent the Flash-Gen modeled distribution, and triangle markers represent the measured distribution, where the plots are in linear scale. Vertical dash-dotted lines are fixed default voltage thresholds.

P/E cycle count, and retention time is used to estimate the conditional probability of that level and time stamp. We visualize the PDFs for measured data and reconstructed data. In addition, we analyze the time variations in the mean and standard deviation values with respect to some program levels.

2. Error count: We compare the generated read voltages with fixed thresholds to determine the read levels and then compute the level error rate for each program level. For quantitative comparison, we compare the read voltages generated by our data-driven method with three statistical fitting methods [78], using the metric of level error rate LER for both the P/E cycling dataset and the retention dataset.

3.5.1 Distribution Analysis

As we evaluate our learned model using input arrays of program levels, we collect regenerated voltage levels and count the frequency of occurrence of voltage levels over the voltage range. We then estimate the PDFs $P(\text{PL} = pl, \text{VL} = vl)$ of voltages associated with each program level and given P/E cycle and retention time.

Fig. 3.6 shows the conditional PDFs for measured data and regenerated data in two evaluation datasets at six different time stamps. The x-axis represents the soft read voltages spanning a certain voltage level range. The y-axis represents the conditional PDF. We only show conditional PDFs from program level 1 to 7 since all the read voltages below the first program threshold are mapped to a fixed value. Despite that fact, we are still able to discuss pattern-dependent errors involving program level 0 in Section 3.6.2.

We obtain three findings from Fig. 3.6. First, our regenerated data (dashed curves) generally fit the measured data (triangle markers) in PDF views and capture the dependence on both temporal factors. The occasional discrepancies in the peak, especially in the plot of 4000 P/E cycles and 0 retention time, do not affect the error analysis of tails in the following parts. The fitted distributions of generated data over higher P/E cycles or larger retention time indicate the precise control of temporal characteristics in Flash-Gen. Second, the peak of the distribution in each program state drops and the distribution becomes wider as the wear condition becomes severe. The wider distributions indicate that more voltage levels exceed the read thresholds and then more errors will be detected. Increasing the number of P/E cycles and increasing the delay between program and read operations lead to an increase in the probability of error. Device lifetime and reliability are both adversely affected. Third, under retention disturbance, the distributions of high program levels shift to lower values of voltage level, and the read voltage distributions of all levels become wider. Similar behaviors exist in 3D TLC flash memory as presented in Fig. 3.2.

In Fig. 3.7, we explore the statistical performance of those measured and learned distributions. We plot the means and standard deviations of distributions $P(\text{PL} = pl, \text{VL} = vl)$ of three program levels as time stamps increase. In all subfigures, we find that trends of Flash-Gen almost match the changes in measured distributions, with some discrepancies in specific values. These matching changes reflect how our Flash-Gen captures the temporal variations in each

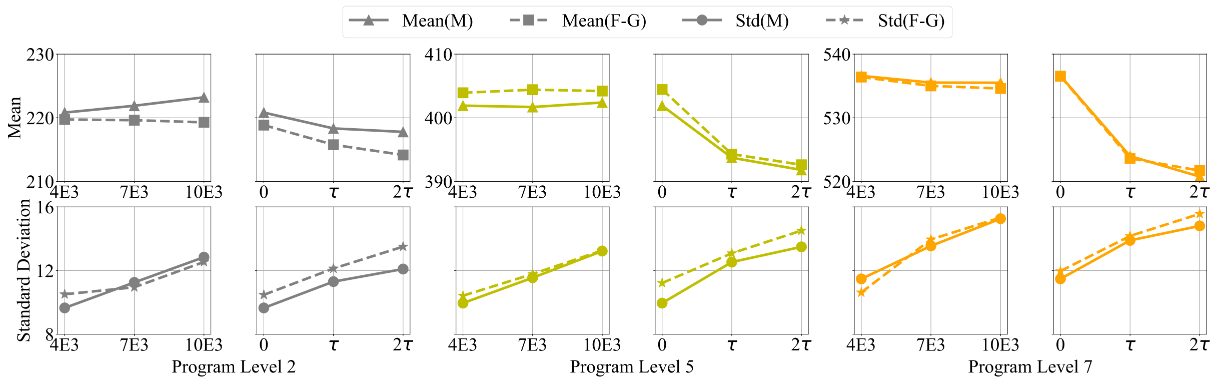

Figure 3.7. Change in mean and standard deviation values for PL at 2, 5, and 7 when P/E cycle count or retention time increases. The first row of subplots presents the mean changes on increasing time stamps and the second row presents the standard deviation values on increasing time stamps. The first two columns are statistics for PL $= 2$, the middle two columns are PL $= 5$, and the last two columns are PL $= 7$. The odd column depicts the variations in P/E cycles and the even column depicts the variations in retention data. In each plot, solid curves with triangle or circle markers are measured data and dashed curves with square or star markers are Flash-Gen data.

program level.

The mean values in the first row of subfigures represent the centers of the distributions for the corresponding program levels and time stamps. For all three program levels, the mean values fall within a voltage interval of width 20, with distinct endpoints for each program level. The mean values remain the same as the P/E cycling increases. However, the average values drop significantly under retention noise, which is consistent with our observations in distribution curves in Fig. 3.6. The mean values in high program levels like PL $= 7$ experience a greater decrease compared to those in low program levels like PL $= 2$. The reason is that the high program levels experience a more rapid leakage of electrons present in the floating gate than the low program levels. As we set a longer retention time, voltages in high program levels leak faster than those in low program levels and we find a larger drop in those mean values.

The standard deviation of each program level reflects the width of each distribution. As the curve becomes wider and wider, more voltage levels will exceed the read thresholds and will be designated as errors. In the second row of Fig. 3.7, we can find that the standard deviation of

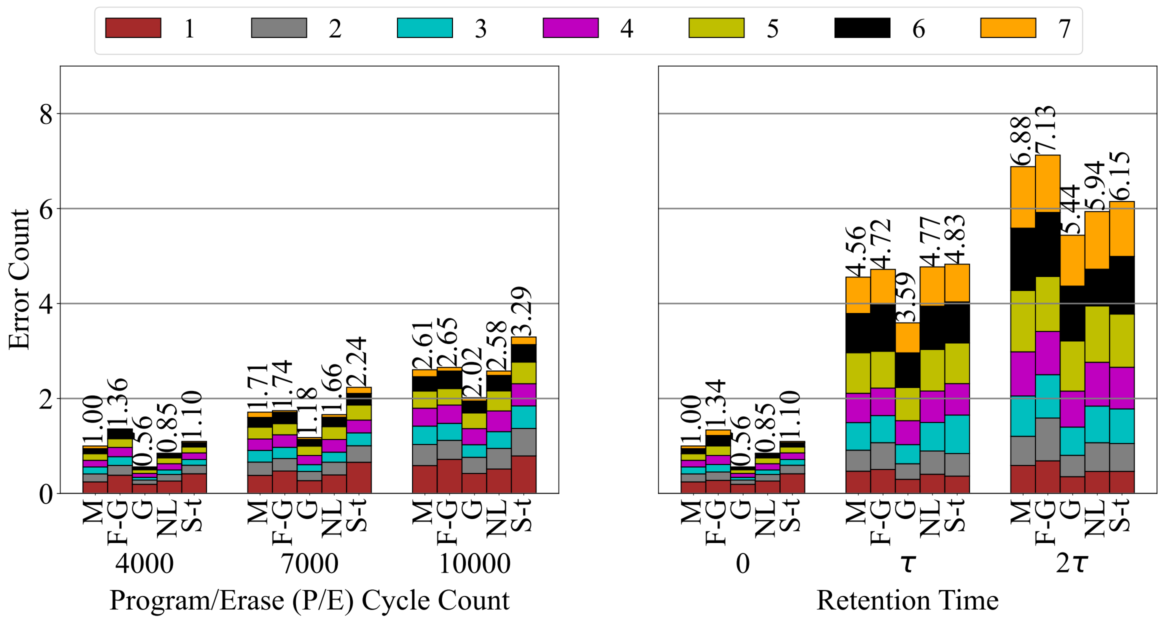

Figure 3.8. Total error rates of measured ('M'), Flash-Gen ('F-G'), Gaussian ('G'), Normal-Laplace ('NL'), Student's t ('S-t') model. The left subfigure is error rates versus P/E cycle count and the right subfigure is error rates versus retention time. In each bar, we stack the errors from program level 1 to program level 7. We normalize the error counts of measured data at 4000 P/E cycle as 1.

each program level increases monotonically with both P/E cycle count and retention time. In the next subsection, we will explore how error rates change as timestamps vary.

3.5.2 Error Rate Estimation

Accurate estimation of error count is essential in the design of wear leveling and ECC algorithms for NAND flash. For a more quantitative assessment of error count, we compare our generative model with measured data and three state-of-the-art statistical models using the metric of error count: Gaussian model [11], Normal-Laplace model [95], and Student's t-distribution model [78]. As far as we know, our proposed machine learning method is the first approach to provide accurate estimation of two-dimensional pattern-dependent and time-dependent errors. Following the optimization process used in [78] for an MLC flash device, we fit those statistical distributions to our TLC measured distributions. We minimize the KL divergence between real distribution P_{real} and fake distribution P_{fake} by using the Nelder-Mead simplex method [90], where the KL divergence is denoted as $D_{KL}(P_{real}, P_{fake})$. We obtain the best-fit parameters for all program levels, except PL $= 0$, with each of the statistical distributions.

We then compute the level error counts under each of the distributions and quantitatively compare those fake distributions with real distributions, where the error count is a measure of the reliability and endurance of the flash memory. To calculate the level error rates from distributions, we fix 7 default read thresholds, as shown by the dash-dotted vertical lines in Fig. 3.6. The hard read voltages are determined by comparing soft voltages to these thresholds. We follow the procedure for the computation of LER in Section 3.4.1 and represent the LER of the five models in Fig. 3.8.

In Fig. 3.8, the x-axis represents the chosen P/E cycle count or retention time, and the label directly under each bar represents the corresponding model name. The y-axis corresponds to the normalized error count. At each timestamp, for each model, the errors from 7 program levels are stacked as one individual bar. The stacked bar represents the total level error rate. We first discuss the measured distributions. Under P/E cycling noise, the total level error rate at 10000 P/E cycles is around $2.5\times$ that at 4000 P/E cycles. LER is increasing when more P/E operations are performed, aligning with our observations in distribution curves and standard deviation. PL $= 1$ has the highest error rate. Under retention noise, the total LER has a substantial increase. LER at τ retention is about $5\times$ larger than that at zero retention and at 2τ retention it is nearly $7\times$ larger than that at zero retention. In the bar plots, high program levels produce more errors than low program levels. These observations can be explained by the shifted and wider distributions in Fig. 3.6. The high error rates observed due to retention distortion confirm that a static set of read thresholds may not be the optimal choice over the entire lifetime of SSDs. These observations on level error rate motivate the investigation of optimal threshold determination and LDPC decoding performance under different P/E cycle counts and retention times.

For the statistical distributions, we observe that the Gaussian model underestimates the error rates in each time stamp; this is because the tails in the actual distribution are becoming heavier as the operating conditions of the device become more severe. The Normal-Laplace model, on the other hand, takes the heavier tails into consideration and provides accurate estimations of error counts at each P/E cycle. Under retention noise, the Normal-Laplace model performs well for retention time τ but does not estimate well for retention time 2τ. Student's t-distribution, the best mathematical model under retention distortion, still underestimates at the retention time 2τ.

For the machine learning approach, Flash-Gen produces more errors than the measured data at 4000 P/E cycles and slightly overestimates the wear conditions in 7000 and 10000 P/E cycles. Flash-Gen can provide accurate estimations of errors under retention distortions, even if the model comes with a slightly higher LER. In conclusion, Normal-Laplace is the best statistical model to capture the distributions of flash devices, except when dealing with high retention time. Our Flash-Gen works better than Normal-Laplace at 7000, 10000 P/E cycles, and τ, 2τ retention time but overestimates the errors at 4000 P/E cycles. A key observation is that the data-driven approach can effectively model channels under severe wear conditions, like 2τ retention time, an advantage not typically presented by mathematical models.

3.6 Performance of Flash-Gen Coding Workflow

In this section, we discuss the performance of the Flash-Gen coding workflow when we employ practical LDPC codes and determine problematic patterns in flash devices. We will present the result of simulations of frame error rates (FERs) under multiple scenarios and pattern-dependent ratios by utilizing our Flash-Gen coding workflow.

3.6.1 LDPC Decoding Performance

In our experiments, we generate around 10 MB of data from our Flash-Gen with selected time stamps and measure about 4 MB of data from the flash chip with more time stamps. We first find the optimal hard thresholds TH_H^* by minimizing BER of generated distributions from

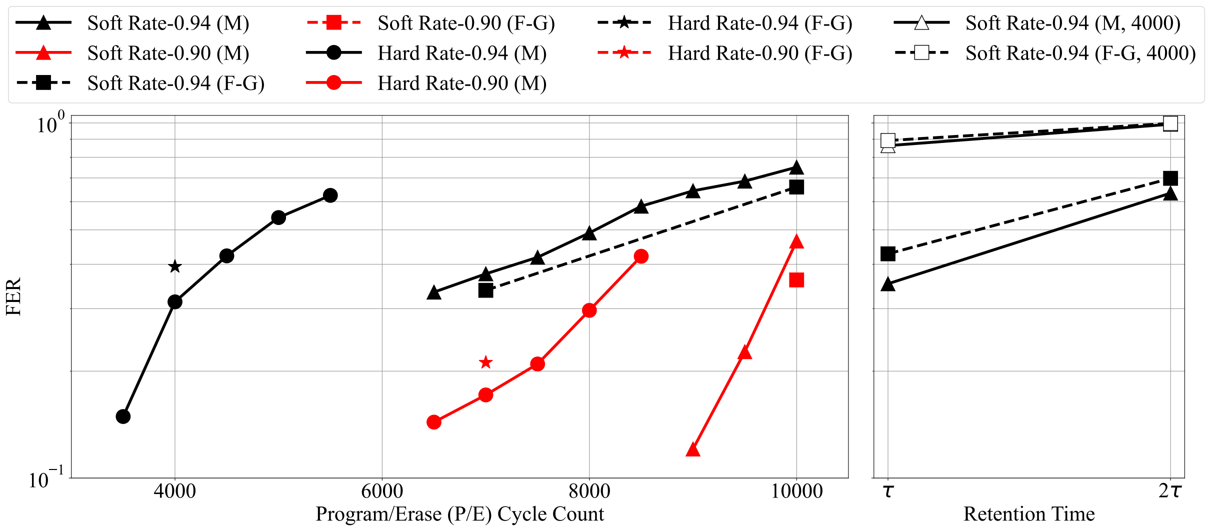

Figure 3.9. Frame error rate (FER) comparison with LDPC codes of measured data and Flash-Gen reconstructed dataset. The time stamp of the left subfigure is P/E cycle count and the time stamp of the right subfigure is retention time. Solid lines correspond to measured data from the flash device and dashed lines present Flash-Gen reconstructed data. Black curves and markers correspond to decoding using rate-0.94 LDPC codes and right curves and markers represent decoding using rate-0.90 LDPC codes. Markers with white fills in the retention plot indicate the sub-optimal thresholds used in decoding.

Flash-Gen and measured distributions from chips. We then select the optimal soft read thresholds TH_S^* and process LLR information for each cell using LLR Processor and LLR Mapper in our LDPC coding procedure. In the following experiments, we use two read thresholds per program level in soft-decision decoding. We then feed the acquired information to the LDPC decoder and observe the decoding performance.

The two LDPC codes are designed based on [46], where the coding rates are approximately 0.94 and 0.90. The length of both codes is 8192. We set the number of decoding iterations to 50. If, after 50 iterations, the LDPC decoder does not decode to the all 0 codeword, we declare a decoding failure. The reconstruction of each cell ensures the authenticity of experiments, setting apart our Flash-Gen from other distribution-based modeling approaches. We map the real or fake voltage levels to their binary representations on three pages and divide each page into several frames based on the code length. The decoder is based upon the belief-propagation decoding algorithm, implemented in software as the floating-point sum-product decoding method.

The FER performance in all three pages regarding measured data and Flash-Gen reconstructed data is shown in Fig. 3.9, where the left figure shows the results for P/E cycling and the right figure shows the results for retention. We first discuss measured data under stressful P/E cycling noise. The rate-0.90 LDPC code performs better than the rate-0.94 code and soft-decision decoding extends the lifetime compared to hard-decision decoding. High-rate codes use fewer parity check bits in error correction and thus provide relatively worse FER performance. The soft-decision decoder utilizes more accurate information regarding voltage levels and produces lower FER compared to hard-decision decoding. Specifically, when FER$= 4 \times 10^{-1}$, rate-0.94 code in soft-decision decoding achieves a gain of about 2800 P/E cycle compared to the same code in hard decoding mode. At the same FER, the rate-0.90 code achieves a gain of about 2700 P/E cycles over the rate-0.94 code under soft-decision decoding. For the rate-0.90 code, soft-decision decoding provides a gain of about 1500 P/E cycles over hard-decision decoding.

We now discuss the FER results under retention distortions. If we optimize soft thresholds TH_S^* using the correct retention distributions, we can obtain slightly better performance using rate-0.94 LDPC codes than soft decoding from 6500 P/E cycles to 10000 P/E cycles. This is consistent with the similar standard deviation in Fig. 3.7. However, when we use inappropriate soft thresholds TH_S $(\text{corresponding to PE} = 4000, \text{retention} = 0)$, we are barely able to decode any frame successfully. When retention time is τ, FER with inappropriate thresholds is almost $2.5\times$ that with correct thresholds. The huge gap in retention FER underscores the critical significance of read thresholds.

The Flash-Gen model trained with data from selected time stamps can accurately estimate the FER. We optimize the thresholds using the learned distributions and feed reconstructions to LDPC decoders. As shown in Fig. 3.9, Flash-Gen produces slightly higher FER than measured data in most experiments, the exception being soft-decision decoding under P/E cycling.

We also studied the effectiveness of Flash-Gen in read threshold optimization using commonly used metrics. We adopt channel BER in (3.9) as the metric in hard-decision threshold determination and MI in (3.11) as the metric in soft-decision threshold determination. Fig. 3.10

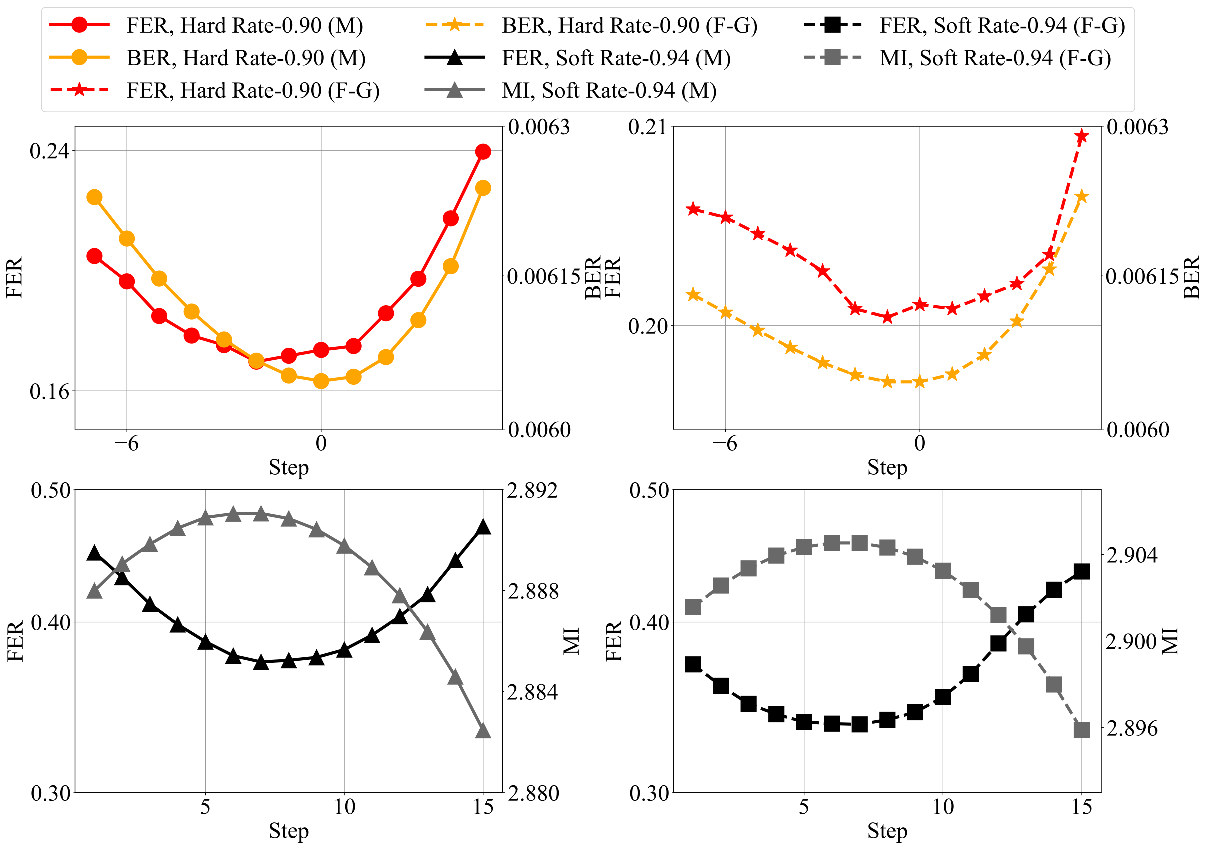

Figure 3.10. Variations in the hard-decision decoding threshold and soft-decision decoding thresholds change the decoding performance and the metric values. The first row shows FER (red) and BER (orange) versus the step change in the hard read threshold position, for measured data (left) and Flash-Gen data (right). The second row shows FER (black) and MI (gray) versus the step change in the soft read threshold positions. In each subplot, the x-axis reflects the step applied to the threshold position(s). The left y-axis is the decoding FER and the right y-axis is the metric value .

illustrates how decoder FER and the respective metrics vary with threshold placement under hard-decision and soft-decision decoding for measured data and Flash-Gen recosntructed data. The "step" represented in the x-axis is interpreted as follows. In the hard experiments, the threshold $\mathrm{TH}_{H,0}$ between PL 0 and PL 1 is moved to position $(\mathrm{TH}_{H,0} + \text{step})$. In the soft experiments, the two read thresholds $\mathrm{TH}_{S,2}$ and $\mathrm{TH}_{S,3}$ between PL 1 and PL 2 are moved to $(\mathrm{TH}^{*}_{H,1} - \text{step})$ and $(\mathrm{TH}^{*}_{H,1} + \text{step})$, respectively.

In the hard-decision decoding experiments, the BER metric will increase as we increase the gap from the optimal threshold. The threshold that generates the lowest FER closely matches

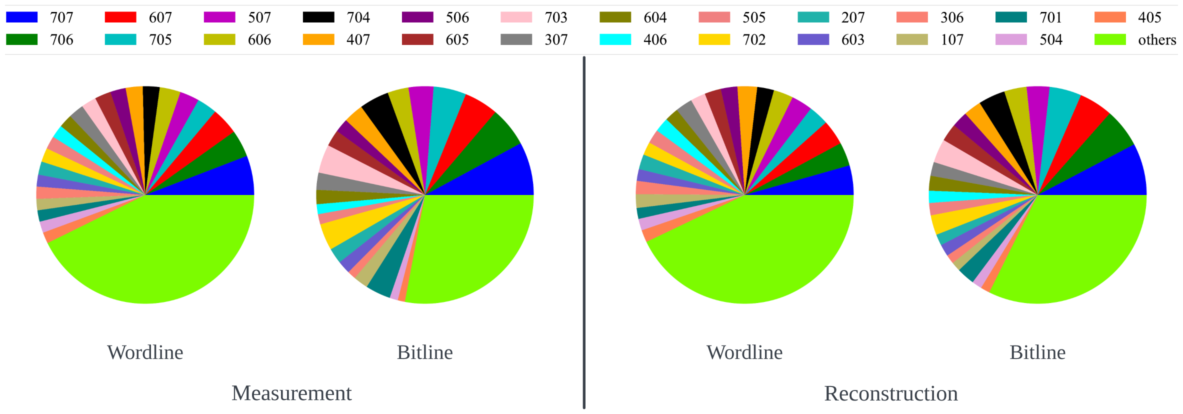

Figure 3.11. Pie charts showing pattern-dependent error probabilities for measured and Flash-Gen generated voltages at 7000 P/E cycles. The sector labeled **others** combines 41 less harmful patterns. In measured data, 97921 total errors are observed at PL $= 0$. In reconstructed data, we sample 10 different latent vectors for inference and 989565 total errors are observed at PL $= 0$.

the optimal threshold for achieving the lowest BER. The Flash-Gen model correctly captures the sensitivity to the variations of threshold placement as reflected in the convex shape of both FER and BER curves. The optimal threshold in the Flash-Gen model leads to the near-optimal decoding FER.

In the soft-decision experiments, the MI curves have similar concave shapes. The step with the highest MI aligns closely with the optimal FER in both the measured plot and the Flash-Gen plot. Since soft-decision decoding requires a longer read time and has a higher decoding complexity than hard-decision decoding, the Flash-Gen model, capable of accomplishing similar tasks, can alleviate the heavy measurement workload on flash controllers.

3.6.2 Design of Constrained Codes

Generating voltage levels with ICI effects is complicated due to pattern-dependent and direction-dependent distortions. We remark that classical statistical models and other machine learning techniques focus on regeneration of the PDFs of the measured data and, as such, are not expected to be effective in capturing ICI effects.

We evaluate how well the generative model learns spatial ICI properties by examining errors associated with program level patterns $\mathrm{PL}_{i,j-1}$ $\mathrm{PL}_{i,j}$ $\mathrm{PL}_{i,j+1}$ and $\mathrm{PL}_{i-1,j}$ $\mathrm{PL}_{i,j}$ $\mathrm{PL}_{i+1,j}$ in the WL and BL directions, respectively. We calculated pattern error probabilities according to (3.15) in Section 3.4.2 in both directions of the P/E cycling dataset. The pattern error probability measures the relative frequency of occurrence of the WL and BL patterns when an error occurs in the victim cell.

The probabilities for measured and regenerated data at 7000 P/E cycle counts are visualized as pie charts in Fig. 3.11. When we program an interior cell to level 0, there are 64 such patterns of program levels for the pair of adjacent cells in both WL and BL directions. Without ICI effects, the errors would occur randomly for all possible patterns. Indeed, we analyze the ICI effects along diagonal directions associated with patterns $\mathrm{PL}_{i-1,j-1}$ $\mathrm{PL}_{i,j}$ $\mathrm{PL}_{i+1,j+1}$ and $\mathrm{PL}_{i-1,j+1}$ $\mathrm{PL}_{i,j}$ $\mathrm{PL}_{i+1,j-1}$. In the 2D TLC flash chip, the ICI effects along diagonal directions are not as severe as along horizontal/vertical directions.

We present 23 dominant patterns as individual sectors and combine other less harmful patterns in one sector. In the measured data, the 23 listed patterns account for 55% of the errors in the WL direction and around 70% of the errors in the BL direction. The dominant error pattern in both WL and BL directions is 707. Comparing the area of pattern 707 in WL and BL, we find that pattern 707 in the WL direction is less severe than that in the BL direction.

As shown in Fig. 3.11, for the prevalent error patterns at 7000 P/E cycles, probabilities observed in the data generated by Flash-Gen with 10 sampled latent vectors during evaluation are very similar to those seen in the measured data. The only substantial discrepancy we observed is that the generative model underestimates the fraction of the pattern 707 in the WL direction. At 4000 (resp., 10000) P/E cycles, the model underestimates (resp., overestimates) the fraction of the 707 pattern in both directions. However, at all P/E cycles, Flash-Gen generates the same rank ordering of pattern fractions as the measured data in both directions.

We prioritize severe ICI patterns in both directions, enabling the design of constrained codes to mitigate ICI effects. As constrained codes lack the ability to correct errors in flash memory, there is an interesting direction of integrating LDPC codes with constrained codes in the future.

3.6.3 Discussion

According to our discussion in Section 3.5 and Section 3.6.2, we demonstrated that the Flash-Gen model can capture complex spatial and temporal characteristics of flash memory devices. Moreover, the Flash-Gen coding workflows provide accurate simulations of flash memory controller operations and provide a helpful tool for analyzing and optimizing the design of LDPC codes and constrained codes.

The proposed Flash-Gen model learns the spatial and temporal properties of flash memory systems when trained on pseudo-random datasets collected from 2D TLC NAND flash devices. However, the learned model cannot be applied to other wear conditions and training the Flash-Gen model from scrath for these other wear conditions requires significant time and resource investments. Fortunately, exploiting transfer learning [130], the learned generative model pre-trained with pseudo-random datasets can be efficiently fine-tuned to code-constrained flash systems using much smaller datasets. Utilizing this transfer learning approach, we have the capability to fine-tune existing models, initially trained on P/E cycling datasets or retention datasets, for use with other target datasets, where the target datasets are from similar domains of the source datasets. The Flash-Gen coding workflows can then be applied to these new models.

In this paper, we analyze P/E cycling errors, retention errors, and ICI effects. Other types of errors occur during the program, erase, and read process, e.g., programming error and read disturb. By collecting appropriate datasets and processing them using the methods in Section 3.3.3, we envision that the data-driven modeling approach can be generalized to model more complex and severe device conditions within flash chips.

3.7 Conclusion

In this paper, we explored the use of conditional generative networks to model the flash memory channel. Unlike traditional modeling and previous machine learning approaches, our model can generate “realistic” soft voltage levels from program level arrays at different time stamps, thereby reflecting both temporal and spatial characteristics of the flash memory channel. We then proposed Flash-Gen coding workflows to support the analysis, design, and performance evaluation of LDPC codes and constrained codes.

We studied the probability distributions and statistical parameters of the read voltages generated by the Flash-Gen model. We then compared the cell-level error rates obtained with these distributions to those of three commonly used mathematical read voltage models. Hard-decision and soft-decision decoding FER results for measured and generated voltages were compared. The sensitivity of these BER and of mutual information metrics as a function of read threshold positioning was also compared. These results demonstrate that the Flash-Gen model accurately represent the spatio-temporal characteristics of flash devices and that Flash-Gen coding workflows can provide realistic simulations of controller functions used in the design and evaluation of ECCs and constrained codes.

Acknowledgement

This chapter is in part a reprint of the material in the paper [126]: Simeng Zheng, Chih-Hui Ho, Wenyu Peng, and Paul H. Siegel, “Flash-Gen: Spatio-temporal generator for flash memory systems,” submitted to *IEEE Transactions on Communications*. The work was presented at DATE 2023 [127]: Simeng Zheng, Chih-Hui Ho, Wenyu Peng, and Paul H. Siegel, “Spatio-temporal modeling for flash memory channels using conditional generative nets,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, Antwerp, Belgium, Apr. 2023. The dissertation author would like to thank the collaborators in this publication.

Chapter 4

Optimal Shaping Codes for a TLC Flash Memory

4.1 Introduction

In some applications, it is useful to model communications and storage channels as a costly channel, a variation of Shannon's discrete noiseless channel where output symbols and sequences of symbols are assigned a positive cost, where the meaning of cost depends on the application. For example, in DNA synthesis, the cost of a sequence of symbols is the number of synthesis cycles required to produce the sequence; in flash memory devices, the cost of a sequence of symbols is the amount of wear inflicted on the programmed cell. Since our goal is to improve the lifetime of TLC flash memory devices, we will view this problem as coding over a costly channel.

For a given finite alphabet $\mathcal{Y}$, any costly channel has a graph representation called a cost graph, which consists of a finite directed graph $G = (V, E)$, an edge labeling function $L : E \to \mathcal{Y}$, and a cost function $\tau : E \to \mathbf{R}^+$ such that there exists a path between any pair of states, the outgoing edges for a given state all have distinct labeling, and τ is non-negative and additive with respect to the edges. As a result, for a given initial state, any path $\gamma = e_1 e_2 ... e_k$ will correspond to the sequence $L(e_1)L(e_2)...L(e_k)$ and will have cost $\sum_{i=1}^{k} \tau(e_i)$.

Since the cost of a sequence directly measures its inflicted wear on a programmed cell of a flash device, the goal of a shaping code is to minimize the average cost per source bit for a given code expansion factor (i.e., inverse code rate). This corresponds to optimally shaping the probability distribution of sequences so that, roughly speaking, high-cost sequences have low probability and low-cost sequences have high probability.

The cost graph used in this study assumes symbol costs that are independent of context; that is, the cost is simply a function of the programmed cell level. Liu *et al.* [70] showed that, in this setting, for an i.i.d. source and a fixed expansion factor f, an optimal shaping scheme can be obtained by concatenating optimal data compression with a code from the compressed data that achieves the overall target expansion factor and minimizes the average cost per compressed bit. They also showed that a fixed-to-variable length Varn code [114] can be used in the second step

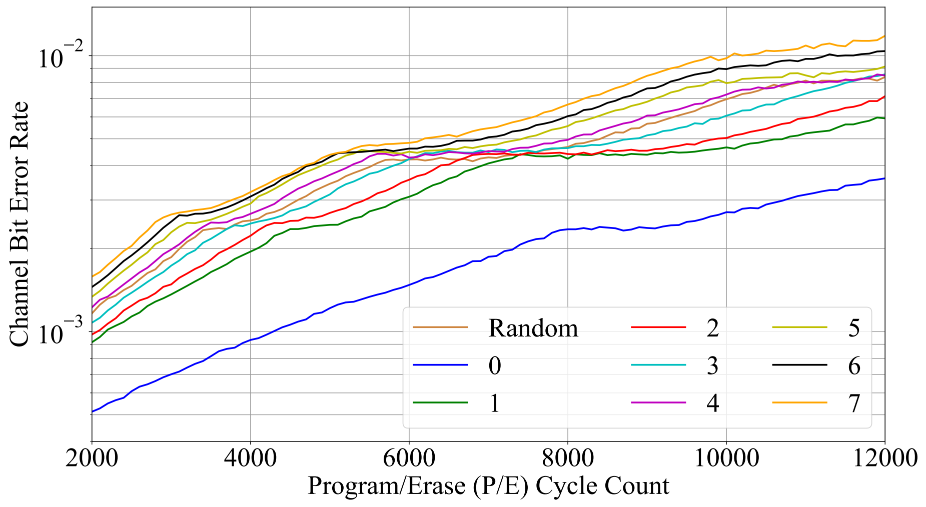

Figure 4.1. Measured BER of pseudorandom data after inducing wear with data that is dominated by a single program level from P/E cycle 2000 to P/E cycle 12000.

to asymptotically achieve optimal shaping.

In [70], optimal shaping schemes with expansion factor $f = 1$ were implemented for a commercial 2y-nm MLC flash device Their performance was evaluated on an English language source (ASCII encoding) and a Chinese language source (UTF-16L encoding), and compared to scenarios with no coding, direct shaping coding [104], [72], and data compression alone. For the English language source, optimal shaping increased P/E cycling lifetime at a bit error rate (BER) of 1×10^{-3} by more than $2.6\times$ over no coding and more than $2.1\times$ over direct shaping. It also allowed the storage of more than $1.15\times$ the number of copies of the source than compression alone. For the Chinese language source, the corresponding gains were about $2.4\times$, $1.9\times$, and $1.25\times$.

In this project, we aim to design and implement optimal shaping codes for TLC flash memories. We give the construction and methodology in Section II and then present our experimental results in Section III. Extensions of the theoretical results in [70] to cost graphs where symbol costs are context-dependent are presented in [71].

4.2 Shaping Codes for TLC Flash

In triple level cell (TLC) NAND flash devices, three bits are stored in each cell to denote one of eight different program (i.e. voltage) levels. This means that the program level of a cell can be represented with an octal symbol, so we use $\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7\}$ as the alphabet for our shaping code.

Based on [8, 72], for each $i \in \mathcal{Y}$ we assign symbol i a constant cost $c_i \geq 0$. Note that the corresponding cost graph will consist of a single state with eight edges, where each symbol and its associated cost is assigned to an edge.

The symbol costs c_i are estimated by performing repeated program and erase (P/E) operations on a TLC flash device and writing almost all cells with one certain program level. For every 100 P/E cycles, we estimate the induced wear of each program level by measuring the corresponding BER; the results are shown in Fig. 4.1, which also includes the BER of random data.

Let BER_{max} denote the maximum tolerable BER. Let T_R be the number of P/E cycles required to achieve BER_{max} when the chip is programmed with random data. For our experiment, we set $BER_{\text{max}} = 2 \times 10^{-3}$. For each $i \in \mathcal{Y}$, let T_i be the number of P/E cycles it takes to reach BER_{max} when inducing wear using only program level i. Then, the cost c_i associated with each level i is defined as $c_i = \frac{T_R}{T_i}$.

We utilize Theorem 2 in [70] to find the level probabilities p_i that minimize the average cost per stored level, $\sum_{i=0}^{7} p_i c_i$, for an overall expansion factor $f = 1$, where the entropy of the source, consisting of Spanish-language texts, is estimated from its compression factor (2.73) under LZ77 compression (gzip). We constructed two Varn codes of codebook sizes 256 and 1024. The symbol costs, c_i, for the 1x-nm TLC flash device, the optimal symbol probabilities, p_i, the relative contribution of each level to the total average cost, $p_i c_i$, and the empirical symbol probabilities achieved by the shaping codes, p_i^{256} and p_i^{1024}, are shown in Table 4.1.

Table 4.1. Symbol Probabilities and Costs for Optimal Shaping Codes

Level	0	1	2	3	4	5	6	7
c_i	0.42	0.76	0.84	0.94	1.03	1.14	1.19	1.28
p_i	0.810	0.085	0.050	0.026	0.014	0.007	0.005	0.003
p_ic_i	0.340	0.065	0.042	0.024	0.014	0.008	0.006	0.004
p_i^{256}	0.777	0.110	0.065	0.025	0.097	0.054	0.049	0.032
p_i^{1024}	0.797	0.089	0.058	0.030	0.016	0.006	0.004	0.001

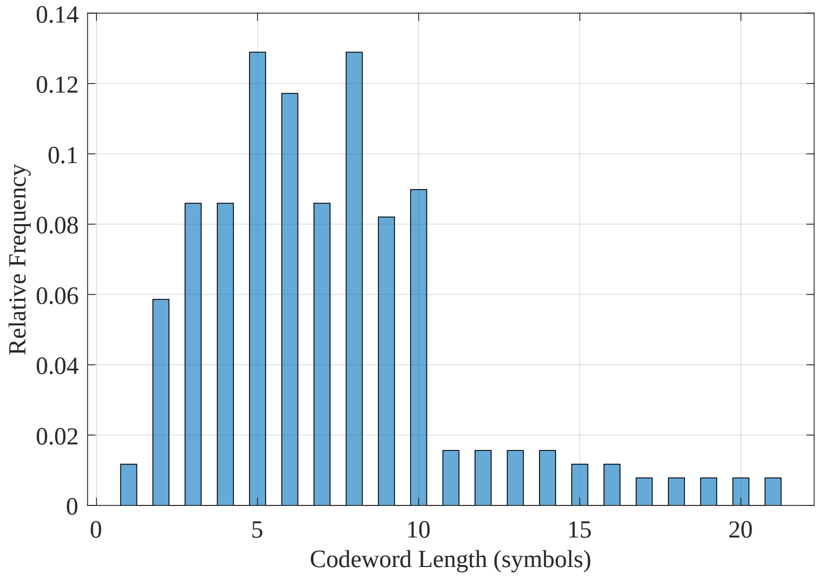

Figure 4.2. Histogram of codeword length in optimal shaping codes with codebook size 256.

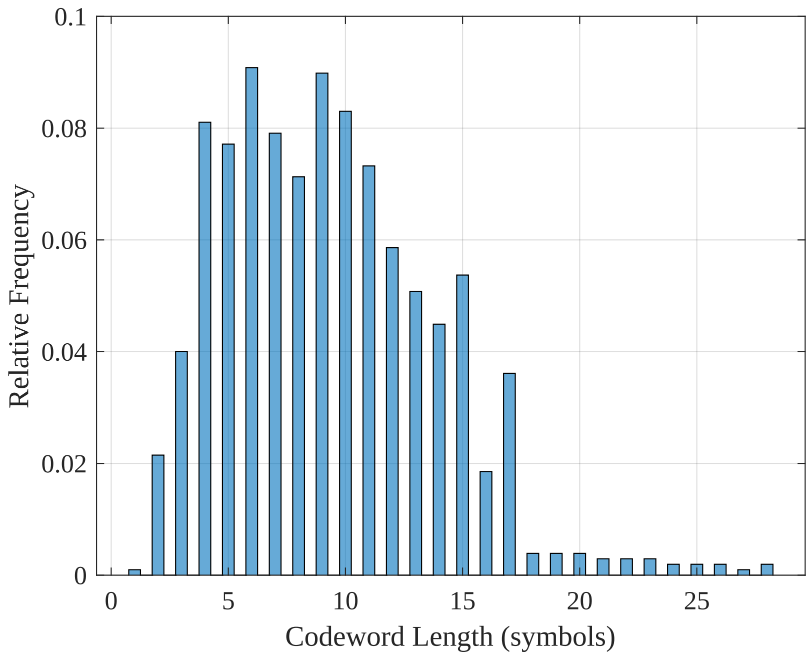

Figure 4.3. Histogram of codeword length in optimal shaping codes with codebook size 1024.

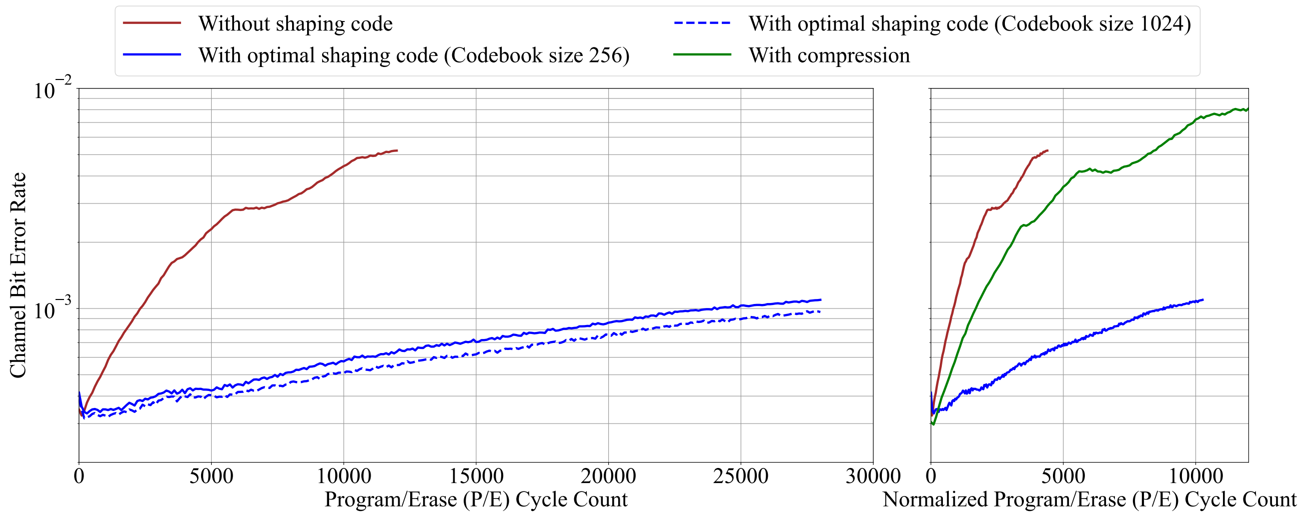

Figure 4.4. (Left) Measured average channel BER comparison when all pages are programmed with Spanish novels data (brown solid), optimal shaping coded data with codebook size 256 (blue solid), optimal shaping coded data with codebook size 1024 (blue dashed) from P/E cycle 0 to P/E cycle 30000. (Right) Measured average channel BER comparison in normalized P/E cycle count including compression data (green solid) from P/E cycle 0 to P/E cycle 12000.

4.3 Experimental Results

The left subfigure of Fig. 4.4 presents the BER performance of the original Spanish-language text and shaping-coded data. When channel BER is 1×10^{-3}, the shaping code with codebook size 256 (resp., 1024) achieves a lifetime gain of $10\times$ (resp., $11.7\times$) over uncoded data. In the right subfigure, we normalize the P/E cycle count by the compression ratio. We see that, compared to using compression alone, the shaping code with codebook size 256 increases the number of times the source text can be written before reaching a BER of 1×10^{-3} by a factor of about $5.2\times$. The histogram of codeword lengths in the Varn code of size 256 is shown in Fig. 4.2 and the histogram of codeword lengths in the Varn code of size 1024 is shown in Fig. 4.3.

A notable observation is that the $10\times$ lifetime gain observed using a Varn code of size 256 for the TLC device is much larger than the $2.6\times$ gain achieved for the MLC device. This is a reflection of the level-dependent wear characteristics of the TLC device, as quantified in

Fig. 4.1. It would be interesting to investigate whether an optimal shaping scheme could achieve even further lifetime gains as flash devices continue to scale up the bit density in each cell.

Acknowledgement

This chapter is in part a reprint of the material in the paper [131]: Simeng Zheng, Andrew Tan, Carolina Fernández, Ismael González Valenzuela, and Paul H. Siegel, "Optimal shaping codes for a TLC flash memory," in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024. The work was presented at Annual Non-Volatile Memories Workshop (NVMW) in Mar. 2024. The dissertation author would like to thank the collaborators in this publication.

Chapter 5

Efficient Constrained Codes That Enable Page Separation in Modern Flash Memories

5.1 Introduction

The history of constrained coding dates back to 1948, when Shannon represented a constrained sequence via a finite-state transition diagram (FSTD) and derived the capacity under a constraint [103]. Run-length-limited (RLL) codes were introduced by Tang and Bahl in 1970 to support the evolution of magnetic recording at that time [111], and these codes were based on lexicographic indexing. In 1973, Cover presented a result about enumerative coding [22] that will prove fundamental for the design of constrained codes based on lexicographic indexing decades later. Among other researchers, Franaszek developed constrained codes based on finite-state machines (FSMs) derived from FSTDs [30]. In 1983, Adler, Coppersmith, and Hassner introduced a systematic method to develop constrained codes based on FSMs [1]. Details about the history of constrained coding until 1998 are in [49].

Because of their ability to improve performance via eliminating error-prone data patterns and undesirable sequences, constrained codes have a plethora of applications. They find application in one-dimensional (1D) magnetic recording devices, both the old ones, which are based on peak detection, and the modern ones, which are based on sequence detection [34, 115]. They can also be combined with robust signal detection using machine learning [128]. They find application in the emerging two-dimensional (2D) magnetic recording devices as well [23, 121]. Moreover, constrained codes are used to achieve DC balance and self-calibration in optical recording devices [47] in addition to many computer standards for data transmission [100]. Recently, constrained codes have been investigated for improved retention and recovery of stored DNA strands in DNA-based storage systems, e.g., [48, 79, 91], and for efficient synthesis of DNA strands [66].

In Flash devices, charge propagation from cells programmed to high charge levels into cells programmed to lower charge levels is the main reason behind inter-cell interference (ICI) [65]. This is correct for any number q of charge levels per cell. Mitigating ICI results in remarkable lifetime gains in Flash as demonstrated in [112] for multi-level cell (MLC) Flash

($q = 4$). There are data patterns that are considered usual suspects for contributing most to ICI. Coding to eliminate data patterns resulting in consecutive levels $(q-1)0(q-1)$ was considered in [88] and [15]. Coding to eliminate data patterns resulting in consecutive levels $(q-1)\mu(q-1)$, also called level patterns, for all $\mu < q-1$, was presented in [112], [15], and [35].

A number of recent results revisited [111] and [22] in order to produce efficient constrained codes based on lexicographic indexing, and one example is [9]. Another example is [34], in which we introduced binary symmetric lexicographically-ordered constrained (S-LOCO) codes and demonstrated density gains in a modern 1D magnetic recording system. We extended our result to single-level cell (SLC) Flash memories ($q = 2$) [33] then to Flash memories with any number q of levels per cell [35]. Moreover, we devised a general method to design LOCO codes for any finite set of patterns to forbid [37], which will be useful in this chapter. We studied the power spectra of binary LOCO codes in [14]. LOCO codes are capacity-achieving, simple, and easily reconfigurable [35, 37].

While the constrained codes in [15] and [35] are quite efficient in terms of rate, they require all Flash pages to be processed together, which negatively affects the access speed. In this chapter, we propose binary *read-and-run (RR)* constrained coding schemes that allow pages to be accessed separately in modern Flash devices, thus preserving high access speed. Our binary RR coding schemes incur small rate loss and work for any Flash device with $q \geq 4$ levels per cell. The key idea is that the constrained code is applied only on one page, the left-most page, while no coding is applied on the other $\log_2 q - 1$ pages. We present a 2D RR coding scheme as well as a 1D RR coding scheme that is based on LOCO codes, and we name the latter binary RR-LOCO coding. Furthermore, we present a 1D 4-ary RR coding scheme that is based on LOCO codes, which we name 4-ary RR-LOCO coding, in order to further reduce the rate loss without impacting the device reliability. In particular, we apply constrained coding on two pages, the two left-most pages, while no coding is applied on the other $\log_2 q - 2$ pages. Therefore, all pages are separated except the two left-most ones. Our 4-ary RR coding scheme works for any Flash device with $q \geq 8$ levels per cell.[1] We show that our 4-ary RR coding scheme can even outperform the binary RR coding schemes at capacity-approaching rates in terms of both complexity and error propagation. There are techniques in the literature that allow page separation; however, they are either incurring notable rate loss [112] or designed for a specific Flash setup [88]. We study various aspects about the proposed RR coding schemes, including charge-level probabilities. We introduce experimental results in a practical triple-level cell (TLC) Flash device ($q = 8$) that demonstrate notable lifetime gains achieved by our coding schemes.

Signal processing techniques have also been proposed to mitigate ICI effects in flash memory. Precompensation, or predistortion, methods [25] attempt to predict the ICI that will be experienced by a cell and modify the program level accordingly. Postcompensation, or postprocessing, methods [3, 25] attempt to estimate the ICI distortion after sensing the cell voltages and apply an appropriate correction to the cell read voltage to offset the ICI effect. Both approaches require accurate information about inter-cell coupling ratios over a range of P/E cycles. They must compute an estimate of the expected ICI for every cell as a function of its program level (or read voltage) and those of its neighboring cells. As pointed out in [3, 25], this comes at a cost of additional processing either during programming or after reading, resulting in extra calculations, additional storage overhead, and added write or read latency. Furthermore, the ICI compensation, whether during programming or after reading, will not be exact, so there may be some residual ICI. On the other hand, results of modeling and simulation in [3, 25] give evidence that these signal processing methods can be very effective in mitigating ICI, and unlike constrained coding methods that introduce redundant data, they do not incur any rate loss penalty.

Given the many issues involved, it is difficult to identify one approach to ICI mitigation as universally superior to another without a careful assessment of the engineering trade-offs. However, further comparison of constrained coding methods to signal processing methods is an interesting topic for future research, as is the consideration of techniques that combine both

[1]This 4-ary RR coding scheme works for $q = 4$ as well, but with more benign patterns forbidden and with no page separation.

methods in a complementary fashion.

5.2 Patterns, Mapping, and 2D RR Coding

As implied in the introduction, literature works do not strictly agree on the set of forbidden patterns to operate on. Additionally, as the Flash device ages, the set of error-prone patterns is expected to expand [35]. According to our recent experimental tests and a machine learning-based ICI characterization [127] of TLC Flash memories, we decided to focus on the set characterized as follows. Let

$$\beta_1, \overline{\beta}_1 \in \mathcal{V}_0 \triangleq \left\{\frac{q}{2}, \frac{q}{2}+1, \dots, q-1\right\}, \tag{5.1}$$

where q is the number of levels per Flash cell (a positive power of 2) and $\mathcal{V}_1 = \{0, 1, \dots, q-1\} \setminus \mathcal{V}_0$. Then, the set of interest is the set resulting in the high-low-high level patterns in $\mathcal{L}_q$:[2]

$$\mathcal{L}_q \triangleq \{\beta_1 \mu \overline{\beta}_1, \forall \beta_1, \overline{\beta}_1 \mid 0 \leq \mu < \min(\beta_1, \overline{\beta}_1)\}. \tag{5.2}$$

This set already subsumes all 3-tuple forbidden patterns adopted in the literature for Flash. This set can be relaxed by removing few patterns that have minimal impact on performance as we shall see in Section 5.4. A block inside the Flash device can be seen as a 2D grid of wordlines and bitlines, with a cell being placed at each intersection [112]. Level patterns in $\mathcal{L}_q$ are detrimental whether they occur on 3 adjacent cells along the same wordline or along the same bitline.

Example 1. *Consider an MLC Flash device, i.e., $q = 4$. In this case, we have $\beta_1, \overline{\beta}_1 \in \{2, 3\}$. Then, the set of interest is the set resulting in:*

$$\mathcal{L}_4 = \{202, 212, 203, 213, 302, 312, 303, 313, 323\}. \tag{5.3}$$

[2]Levels are defined through their indices $\{0, 1, \dots, q-1\}$ for simplicity.

The last three elements in $\mathcal{L}_4$ are quite known [35, 88, 112].

Algorithm 1. Recursive Alternate Gray Mapping

```
1: Input: Number of levels per cell $q$, and $p = \log_2 q$.
2: Define map, a binary array of dimensions $q \times p$.
3: Set $\mathrm{map}(0, :) = \mathbf{1}^p$. (a sequence of $p$ 1's)
4: for $i \in \{0, 1, \dots, p-1\}$ do
5:     for $j \in \{0, 1, \dots, 2^i - 1\}$ do
6:         $\mathrm{map}(2^i + j, :) = \mathrm{map}(2^i - 1 - j, :)$.
7:         Flip the bit $\mathrm{map}(2^i + j, i)$. (each sequence in map is indexed from right to left by $0, 1, \dots, p-1$)
8:     end for
9: end for
10: Output: Array map that maps each index to binary data.
```

Next, we discuss how to map from data to charge levels in Flash and vice versa. Since we are interested in page separation throughout this work, the mapping here is from a charge level out of q possible ones to $\log_2 q$ binary bits, one for each page, and vice versa. Gray mapping offers the advantage that there is only one-bit difference between any two adjacent charge levels, which is valuable for error performance. We adopt a recursive alternate Gray mapping (RAGM), and Algorithm 1 shows how to produce it for any $q \geq 4$. We highlight that RAGM has already been used in the literature in MLC Flash [112] and TLC Flash [88]. Thus, RAGM is not strictly a new contribution.

Example 2. *Consider a TLC Flash device, i.e., $q = 8$. In this case, the output of Algorithm 1, which is RAGM, becomes:*

$$\begin{aligned}
&0 \longleftrightarrow 111, \qquad && 1 \longleftrightarrow 110, \\
&2 \longleftrightarrow 100, && 3 \longleftrightarrow 101, \\
&4 \longleftrightarrow 001, && 5 \longleftrightarrow 000, \\
&6 \longleftrightarrow 010, && 7 \longleftrightarrow 011.
\end{aligned} \tag{5.4}$$

Now, we are ready to discuss binary coding schemes. Let us first index the Flash pages the same way the bits in each sequence in the array map are indexed (see Algorithm 1). This means that the left-most page is the one indexed by $p-1$. From (5.2) and Algorithm 1, the level patterns in $\mathcal{L}_q$ correspond to binary patterns where the left-most page (pages) always has (have) two 0's separated by some bit, i.e., $0x0$. Based on that, forbidding $\{000, 010\}$ on the left-most page (pages) guarantees that no level pattern in $\mathcal{L}_q$ would appear while writing to a Flash device, with any $q \geq 4$, at least in the wordline (bitline) direction. This corresponds to an interleaved RLL $(d, k) = (0, 1)$ constraint [107]. Notably, no coding on any other page is needed. Data will therefore be read from each page independently, and immediately passed to the low-density parity-check (LDPC) decoder to start its processing. This idea is the key idea of our binary RR constrained coding schemes.[3]

RR coding can be performed in the wordline direction only (1D), the bitline direction only (1D), or both directions (2D). Observe that binary RR coding will also prevent some benign level patterns, e.g., 474, 555, and 676 in TLC Flash, resulting in inevitable rate loss. However, as we shall see in Section 5.5, this rate loss is small. Furthermore, some of these benign level patterns will be allowed when we shift from binary to 4-ary coding, which reduces this rate loss, as we shall see in Section 5.4. RR-LOCO codes are capacity-approaching codes.

We start here with our scheme for 2D binary RR constrained coding. As the name suggests, we want to prevent the patterns in $\mathcal{R}^2 = \{000, 010\}$ from appearing at the left-most pages in both wordline and bitline directions in the Flash device through simple encoding and decoding. The encoding follows the rules:

1. On wordlines with indices congruent to 0 or 1 (mod 4), you are allowed to write 0's and 1's freely in bit positions congruent to 0 or 1 (mod 4) at the left-most pages.

2. On wordlines with indices congruent to 2 or 3 (mod 4), you are allowed to write 0's and 1's freely in bit positions congruent to 2 or 3 (mod 4) at the left-most pages.

3. In the other bit positions, you can only write 1's on wordlines at the left-most pages.

[3]An equivalent scheme was proposed for MLC Flash, i.e., $q = 4$, in [107].

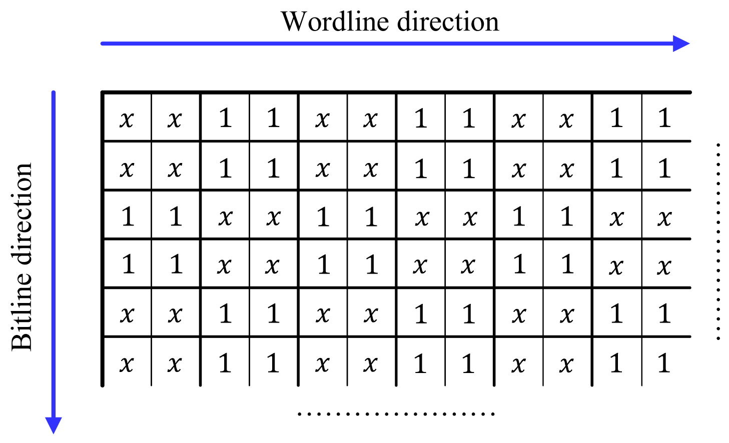

Figure 5.1. The left-most pages of a 2D Flash grid with data encoded via the proposed 2D binary RR coding scheme. Symbol x means bit can be 0 or 1 freely.

This 2D binary RR constrained coding scheme is depicted in Fig. 5.1. It is clear from the figure that the patterns in $\mathcal{R}^2 = \{000, 010\}$ are eliminated on the left-most pages, which forbids all level patterns in $\mathcal{L}_q$, in both directions. Upon encoding, input data bits are freely placed at the positions marked by x for the left-most pages, and they are directly placed (uncoded) at the other pages. Upon decoding, information at the positions marked by 1 is omitted, and data bits at the remaining positions are read with no additional processing and with no correlation between different Flash pages.[4]

This 2D binary scheme is ideal in terms of complexity, access speed, and error propagation (see Section 5.5). It might also seem notably better than any 1D scheme (binary or 4-ary) in terms of performance. However, 1D schemes can achieve almost the same performance with higher rates, which we will discuss in more detail later.

[4]An equivalent 2D scheme forbidding patterns $\{101, 111\}$ on the right-most pages in both wordline and bitline directions in MLC Flash was proposed in [107].

5.3 RR-LOCO Coding Over GF(2)

In this section, we introduce a binary RR coding scheme that forbids $\{000, 010\}$ on the left-most pages in either the wordline direction or the bitline direction, while leaving all other pages with no coding, which forbids the level patterns in $\mathcal{L}_q$ and achieves page separation. This scheme is the binary RR-LOCO coding scheme. The constrained code we apply is a binary LOCO code devised according to the general method in [37]. We start by defining the proposed LOCO code.

Definition 1. *A binary LOCO code $\mathcal{RC}_m^2$, where $m \geq 1$, that forbids the patterns in $\mathcal{R}^2 = \{000, 010\}$ is defined by the following properties:*

1. *Codewords in $\mathcal{RC}_m^2$ are defined over GF$(2) = \{0, 1\}$ and are of length m bits.*
2. *Codewords in $\mathcal{RC}_m^2$ are ordered lexicographically.*
3. *Codewords in $\mathcal{RC}_m^2$ do not have patterns in $\mathcal{R}^2$.*
4. *All codewords satisfying 1)–3) are included.*

Lexicographic ordering is ordering codewords ascendingly according to the rule "$0 < 1$", where bit significance reduces from left to right [35, 111]. The first step to devise this binary LOCO code is to specify the group structure. Codewords in $\mathcal{RC}_m^2$, $m \geq 2$, can be partitioned into the following groups:

- Group 1: Codewords starting with 0011 from the left.
- Group 2: Codewords starting with 011 from the left.
- Group 3: Codewords starting with 1 from the left.

The second step is to enumerate the codewords in $\mathcal{RC}_m^2$, which is done by Theorem 1. Let $N_2(m) \triangleq |\mathcal{RC}_m^2|$.

Theorem 1. *The cardinality of a binary LOCO code $\mathcal{RC}_m^2$ is given by the recursive formula:*

$$N_2(m) = N_2(m-1) + N_2(m-3) + N_2(m-4),\ m \geq 2, \tag{5.5}$$

where the defined cardinalities are:

$$N_2(-3) \triangleq 0,\ N_2(-2) = N_2(-1) = N_2(0) \triangleq 1,\ \textit{and}\ N_2(1) = 2. \tag{5.6}$$

Proof. We compute the cardinalities of each group then add them all. Let the cardinality of Group i be $N_{2,i}$. As for Group 3 in $\mathcal{RC}_m^2$, there is a bijection between its codewords and the codewords in $\mathcal{RC}_{m-1}^2$ (attach 1 from the left). Thus,

$$N_{2,3}(m) = N_2(m-1). \tag{5.7}$$

As for Group 2 in $\mathcal{RC}_m^2$, there is a bijection between its codewords and the codewords starting with 1 from the left in $\mathcal{RC}_{m-2}^2$ (attach 01 from the left). Thus using (5.7),

$$N_{2,2}(m) = N_{2,3}(m-2) = N_2(m-3). \tag{5.8}$$

As for Group 1 in $\mathcal{RC}_m^2$, there is a bijection between its codewords and the codewords starting with 1 from the left in $\mathcal{RC}_{m-3}^2$ (attach 001 from the left). Thus using (5.7),

$$N_{2,1}(m) = N_{2,3}(m-3) = N_2(m-4). \tag{5.9}$$

Adding (5.7), (5.8), and (5.9) gives (5.5). The defined cardinalities, other than $N_2(1)$, can be computed by observing that $N_2(1) = 2$, $N_2(2) = 4$, $N_2(3) = 6$, and $N_2(4) = 9$, which sets up four equations. This observation is immediate given the forbidden patterns. ■

Define a codeword $\mathbf{c}$ in $\mathcal{RC}_m^2$ as $\mathbf{c} \triangleq c_{m-1}c_{m-2}\dots c_0$, with $c_i \triangleq \zeta$ for $i \geq m$, where

ζ represents "out of codeword bounds". The integer equivalent of a LOCO codeword bit c_i, $0 \leq i \leq m-1$, is a_i, i.e., a_i is 0 (1) when c_i is 0 (1). Denote the lexicographic index of a codeword $\mathbf{c}$ among all codewords in the LOCO code $\mathcal{RC}_m^2$ by $g_2(m, \mathbf{c})$, which is abbreviated to $g(\mathbf{c})$. In general, $g(\mathbf{c})$ is in $\{0, 1, \dots, N_2(m) - 1\}$.

The third step is to specify the special cases of occurence for a 1 inside a codeword in $\mathcal{RC}_m^2$. These cases are:

- Case 2: $c_{i+2}c_{i+1}c_i = 001$.
- Case 3: $c_{i+2}c_{i+1}c_i = 011$.
- Case 4: $c_{i+2}c_{i+1}c_i = 101$ or $c_{i+2}c_{i+1}c_i = \zeta 01$.

The typical or default case, Case 1, is simply the case of "otherwise". In particular, it is the case that $c_{i+2}c_{i+1}c_i = 111$, $c_{i+2}c_{i+1}c_i = \zeta 11$, or $c_{i+1}c_i = \zeta 1$.

The fourth and fifth steps are to find the encoding-decoding rule, which specifies the mapping from index to codeword and vice versa. This rule for $\mathcal{RC}_m^2$ is given in Theorem 2.

Theorem 2. *The relation between the lexicographic index $g(\mathbf{c})$, $\mathbf{c} \in \mathcal{RC}_m^2$, and the binary codeword $\mathbf{c}$ itself is given by:*

$$g(\mathbf{c}) = \sum_{i=0}^{m-1} a_i \Big[(1 - y_{i,1}) N_2(i-2) + (1 - y_{i,1} - y_{i,2}) N_2(i-3) \Big], \tag{5.10}$$

where $y_{i,1}$ and $y_{i,2}$ are specified as follows:

$$\begin{aligned} &y_{i,1} = 1 \textit{ if } c_{i+2}c_{i+1}c_i \in \{001, 011\}, \textit{ and } y_{i,1} = 0 \textit{ otherwise}, \\ &y_{i,2} = 1 \textit{ if } c_{i+1}c_i = 01 \text{ s.t. } y_{i,1} = 0, \textit{ and } y_{i,2} = 0 \textit{ otherwise}. \end{aligned} \tag{5.11}$$

Proof. We compute the contributions $g_{i,j}(c_i)$ of a bit c_i under Case j, for all $j \in \{1, 2, 3, 4\}$, in a binary LOCO codeword then merge them all. As for the typical case, which we index by 1, this contribution is the number of codewords starting with 0 from the left in $\mathcal{RC}^2_{i+1}$. Thus using (5.8) and (5.9),

$$\begin{aligned} g_{i,1}(c_i) &= N_{2,2}(i+1) + N_{2,1}(i+1) \\ &= N_2(i-2) + N_2(i-3). \end{aligned} \tag{5.12}$$

As for Case 2 (Case 3), this contribution is the number of codewords starting with 000 (010) from the left in $\mathcal{RC}^2_{i+3}$. Note that 000 and 010 are forbidden patterns. Thus,

$$\begin{aligned} &g_{i,2}(c_i) = 0 \text{ and} \\ &g_{i,3}(c_i) = 0. \end{aligned} \tag{5.13}$$

As for Case 4, this contribution is the number of codewords starting with 00 from the left in $\mathcal{RC}^2_{i+2}$. Thus using (5.9),

$$g_{i,4}(c_i) = N_{2,1}(i+2) = N_2(i-2). \tag{5.14}$$

Using $y_{i,1}$ (for Cases 2 and 3) and $y_{i,2}$ (for Case 4) from (5.11) along with a_i to merge (5.12), (5.13), and (5.14) gives:

$$g_i(c_i) = a_i\Big[(1-y_{i,1})N_2(i-2) + (1-y_{i,1}-y_{i,2})N_2(i-3)\Big]. \tag{5.15}$$

Substituting (5.15) in $g(\mathbf{c}) = \sum_{i=0}^{m-1} g_i(c_i)$ gives (5.10). ■

For brevity, we skip the sixth step, which is to assemble the encoding and decoding algorithms. These algorithms are a direct consequence of the rule in (5.10), and we refer the reader to [111], [35], [37], and [63] for details. Note that we sometimes refer to $\mathcal{RC}^2_m$ as a *1D binary RR-LOCO code*. The encoding-decoding rule of a LOCO code is the reason behind its low complexity algorithms, where reconfiguration becomes as easy as reprogramming an adder [34, 37].

Remark 2. *If the coded bits are complemented before writing to pages, the set of forbidden patterns on the left-most pages becomes* $\{101, 111\}$ *instead, which appears in [112] as well. In this case, the cardinality of the binary LOCO code remains as in (5.5), while the encoding-decoding rule becomes exactly that of a binary asymmetric LOCO code in [33] for* $x = 1$*:*

$$g(\mathbf{c}) = \sum_{i=0}^{m-1} a_i N_2(i - a_{i+1}). \tag{5.16}$$

Encoding and decoding on the left-most pages are just subtractions and additions. As for the remaining pages, data is written and read directly (uncoded). This guarantees simplicity and maintains high access speed via our 1D binary RR-LOCO coding scheme.

5.4 RR-LOCO Coding Over GF(4)

In this section, we propose a 1D RR coding scheme over GF(4), which is also based on LOCO codes. This scheme is our 4-ary RR-LOCO coding scheme. The goal is to limit the rate loss resulting from binary RR coding schemes via coding on the two left-most pages. Finer classification of error-prone patterns, stemming from characterizing them via two bits instead of one, results in allowing some benign or less detrimental patterns, and therefore increasing the rate with negligible effect on performance.

We start by modifying the set of error-prone patterns. Let

$$\begin{aligned}
\theta_1, \overline{\theta}_1 \in \mathcal{W}_0 &\triangleq \left\{\frac{3q}{4}, \frac{3q}{4}+1, \dots, q-1\right\}, \\
\theta_2, \overline{\theta}_2 \in \mathcal{W}_1 &\triangleq \left\{\frac{q}{2}, \frac{q}{2}+1, \dots, \frac{3q}{4}-1\right\}, \\
\theta_3, \in \mathcal{W}_2 \cup \mathcal{W}_3,\ \mathcal{W}_2 &\triangleq \left\{\frac{q}{4}, \frac{q}{4}+1, \dots, \frac{q}{2}-1\right\},\ \mathcal{W}_3 \triangleq \left\{0, 1, \dots, \frac{q}{4}-1\right\},
\end{aligned} \tag{5.17}$$

where q is the number of levels per Flash cell (a positive power of 2). While mathematically

$q \geq 4$, we focus here on the case of $q \geq 8$. Then, the set of interest is the set resulting in the high-low-high level patterns in $\mathcal{L}'_q \subset \mathcal{L}_q$:

$$\mathcal{L}'_q \triangleq \{\theta_1 \eta \overline{\theta}_1, \forall \theta_1, \overline{\theta}_1 \mid 0 \leq \eta < \min(\theta_1, \overline{\theta}_1)\} \cup \{\theta_1 \theta_3 \theta_2, \forall \theta_1, \theta_2, \theta_3\} \cup \\ \{\theta_2 \theta_3 \theta_1, \forall \theta_1, \theta_2, \theta_3\} \cup \{\theta_2 \theta_3 \overline{\theta}_2, \forall \theta_2, \overline{\theta}_2, \theta_3\}. \tag{5.18}$$

This set also subsumes all 3-tuple forbidden patterns adopted in the literature for Flash. The only difference between the set $\mathcal{L}'_q$ and the set $\mathcal{L}_q$ is that in the former, if either the left level is or the right level is or both levels are in $\mathcal{W}_1$, the middle level is always in $\mathcal{W}_2 \cup \mathcal{W}_3$. Our experimental results show that the level patterns in $\mathcal{L}_q \setminus \mathcal{L}'_q$ have very limited contribution to the errors occurring upon reading from the Flash device.

Example 3. *Consider a TLC Flash device, i.e., $q = 8$. In this case, we have $\theta_1, \overline{\theta}_1 \in \{6, 7\}$, $\theta_2, \overline{\theta}_2 \in \{4, 5\}$, and $\theta_3 \in \{0, 1, 2, 3\}$. Then, the difference between the two sets of interest is only one level pattern:*

$$\mathcal{L}_8 \setminus \mathcal{L}'_8 = \{545\}. \tag{5.19}$$

For mapping from charge levels to binary bits, we adopt the RAGM of Algorithm 1. Moreover, we index the Flash pages the same way the bits in each sequence in the array map are indexed using Algorithm 1. Therefore, we are interested here in the data on the two left-most pages indexed by $p - 1$ and $p - 2$. We adopt the following binary to 4-ary mapping-demapping, where GF$(4) = \{0, 1, \alpha, \alpha^2\}$, for these two specific Flash pages:

$$11 \longleftrightarrow 0 \ (\mathcal{W}_3), \qquad 10 \longleftrightarrow 1 \ (\mathcal{W}_2), \\ 00 \longleftrightarrow \alpha \ (\mathcal{W}_1), \qquad 01 \longleftrightarrow \alpha^2 \ (\mathcal{W}_0). \tag{5.20}$$

The set of level patterns corresponding to each GF(4) symbol is given between parenthesis.

We can see from (5.17), (5.18), and (5.20) that the set of level patterns in $\mathcal{L}'_q$ can be forbidden in the wordline or the bitline direction by forbidding the 4-ary patterns in the following set $\mathcal{R}^4$ from being written on the two left-most pages indexed by $p-1$ and $p-2$:

$$\mathcal{R}^4 = \{\alpha 0\alpha, \alpha 1\alpha, \alpha 0\alpha^2, \alpha 1\alpha^2, \alpha^2 0\alpha, \alpha^2 1\alpha, \alpha^2 0\alpha^2, \alpha^2 1\alpha^2, \alpha^2\alpha\alpha^2, \alpha^2\alpha^2\alpha^2\}. \tag{5.21}$$

Once again, no coding on any other page is needed. Data will therefore be read from each page independently, except the two left-most pages, and immediately passed to the low-density parity-check (LDPC) decoder to start its processing. This idea is the key idea of our 4-ary RR constrained coding scheme.

Consider a TLC Flash device ($q = 8$) once again. Forbidding the patterns in $\mathcal{R}^4$ on the two left-most pages instead of the patterns in $\mathcal{R}^2$ on the left-most page results in allowing many benign patterns that are forbidden if binary RR coding is adopted, e.g., 444, 474, and 555.

Now, we introduce our 4-ary RR coding scheme that forbids the patterns in $\mathcal{R}^4$ on the two left-most pages in either the wordline direction or the bitline direction, while leaving all other pages with no coding. The constrained code we apply is a 4-ary LOCO code devised according to the general method in [37]. We start by defining the proposed LOCO code.

Definition 2. *A 4-ary LOCO code $\mathcal{RC}^4_m$, where $m \geq 1$, that forbids the patterns in $\mathcal{R}^4$ is defined by the following properties:*

1. *Codewords in $\mathcal{RC}^4_m$ are defined over GF$(4) = \{0, 1, \alpha, \alpha^2\}$ and are of length m symbols.*
2. *Codewords in $\mathcal{RC}^4_m$ are ordered lexicographically.*
3. *Codewords in $\mathcal{RC}^4_m$ do not have patterns in $\mathcal{R}^4$.*
4. *All codewords satisfying 1)–3) are included.*

Lexicographic ordering here is ordering codewords ascendingly according to the rule "$0 < 1 < \alpha < \alpha^2$", where symbol significance reduces from left to right [35, 111]. The first step to devise this 4-ary LOCO code is to specify the group structure. Let γ_1 and γ_2 be in $\{0,1\}$. Codewords in $\mathcal{RC}_m^4$, $m \geq 3$, can be partitioned into the following groups:

- Group 1: Codewords starting with γ_1, $\forall \gamma_1$, from the left.
- Group 2: Codewords starting with $\alpha\gamma_1\gamma_2$, $\forall \gamma_1, \gamma_2$, from the left.
- Group 3: Codewords starting with $\alpha\alpha$ or $\alpha\alpha^2$ from the left.
- Group 4: Codewords starting with $\alpha^2\gamma_1\gamma_2$, $\forall \gamma_1, \gamma_2$, from the left.
- Group 5: Codewords starting with $\alpha^2\alpha\gamma_1\gamma_2$, $\forall \gamma_1, \gamma_2$, from the left..
- Group 6: Codewords starting with $\alpha^2\alpha\alpha$ from the left.
- Group 7: Codewords starting with $\alpha^2\alpha^2\gamma_1\gamma_2$, $\forall \gamma_1, \gamma_2$, from the left..
- Group 8: Codewords starting with $\alpha^2\alpha^2\alpha$ from the left.

The second step is to enumerate the codewords in $\mathcal{RC}_m^4$, which is done by Theorem 3. Let $N_4(m) \triangleq |\mathcal{RC}_m^4|$.

Theorem 3. *The cardinality of a* 4*-ary LOCO code* $\mathcal{RC}_m^4$ *is given by the recursive formula:*

$$\begin{aligned} N_4(m) = {} & 3N_4(m-1) - 2N_4(m-2) + 9N_4(m-3) \\ & + 7N_4(m-4) + 6N_4(m-5) + 4N_4(m-6),\ m \geq 3, \end{aligned} \tag{5.22}$$

where the defined cardinalities are:

$$\begin{gathered} N_4(-5) \triangleq \frac{1}{32},\ N_4(-4) \triangleq -\frac{1}{16},\ N_4(-3) \triangleq 0,\ N_4(-2) \triangleq \frac{1}{4},\ N_4(-1) \triangleq \frac{1}{2},\ N_4(0) \triangleq 1, \\ \textit{and } N_4(1) = 4,\ N_4(2) = 16. \end{gathered} \tag{5.23}$$

Proof. We compute the cardinalities of each group then add them all. Let the cardinality of Group i be $N_{4,i}$. As for Group 1 in $\mathcal{RC}_m^4$, there is a surjection between its codewords and the codewords in $\mathcal{RC}_{m-1}^4$ (attach 0 or 1 from the left). Thus,

$$N_{4,1}(m) = 2N_4(m-1). \tag{5.24}$$

As for Group 2 in $\mathcal{RC}_m^4$, there is a surjection between its codewords and the codewords in $\mathcal{RC}_{m-3}^4$. Thus,

$$N_{4,2}(m) = (2)(2)N_4(m-3) = 4N_4(m-3). \tag{5.25}$$

As for Group 3 in $\mathcal{RC}_m^4$, there is a bijection between its codewords and the codewords starting with α or α^2 from the left in $\mathcal{RC}_{m-1}^4$. Thus using (5.24),

$$N_{4,3}(m) = N_4(m-1) - N_{4,1}(m-1) = N_4(m-1) - 2N_4(m-2). \tag{5.26}$$

As for Group 4 in $\mathcal{RC}_m^4$, the cardinality is the same as that of Group 2. Thus,

$$N_{4,4}(m) = (2)(2)N_4(m-3) = 4N_4(m-3). \tag{5.27}$$

As for Group 5 in $\mathcal{RC}_m^4$, it is handled in a way similar to that of Groups 2 and 4. Thus,

$$N_{4,5}(m) = (2)(2)N_4(m-4) = 4N_4(m-4). \tag{5.28}$$

As for Group 6 in $\mathcal{RC}_m^4$, there is a bijection between its codewords and the codewords starting with α from the left in $\mathcal{RC}_{m-2}^4$. Thus using (5.25) and (5.26),

$$N_{4,6}(m) = N_{4,2}(m-2) + N_{4,3}(m-2) = N_4(m-3) - 2N_4(m-4) + 4N_4(m-5). \tag{5.29}$$

As for Group 7 in $\mathcal{RC}_m^4$, the cardinality is the same as that of Group 5. Thus,

$$N_{4,7}(m) = (2)(2)N_4(m-4) = 4N_4(m-4). \tag{5.30}$$

As for Group 8 in $\mathcal{RC}_m^4$, there is a bijection between its codewords and the codewords starting with $\alpha^2\alpha$ from the left in $\mathcal{RC}_{m-1}^4$. Thus using (5.28) and (5.29),

$$N_{4,8}(m) = N_{4,5}(m-1) + N_{4,6}(m-1) = N_4(m-4) + 2N_4(m-5) + 4N_4(m-6). \tag{5.31}$$

Adding (5.24), (5.25), (5.26), (5.27), (5.28), (5.29), (5.30), and (5.31) gives (5.22). The defined cardinalities, other than $N_4(1)$ and $N_4(2)$, can be computed from the cardinalities at small values of m, which set up six equations. ■

Define a codeword $\mathbf{c}$ in $\mathcal{RC}_m^4$ as $\mathbf{c} \triangleq c_{m-1}c_{m-2}\dots c_0$, with $c_i \triangleq \zeta$ for $i \geq m$, where ζ represents "out of codeword bounds". The integer equivalent of a LOCO codeword symbol c_i, $0 \leq i \leq m-1$, is a_i, i.e., a_i is 0, 1, 2, or 3 when c_i is 0, 1, α, or α^2, respectively. Denote the lexicographic index of a codeword $\mathbf{c}$ among all codewords in the LOCO code $\mathcal{RC}_m^4$ by $g_4(m, \mathbf{c})$, which is abbreviated to $g(\mathbf{c})$. In general, $g(\mathbf{c})$ is in $\{0, 1, \dots, N_4(m) - 1\}$.

The third step is to specify the typical/special cases of occurence for a symbol in GF$(4) \setminus \{0\}$ inside a codeword in $\mathcal{RC}_m^4$. Let γ be in $\{\zeta, 0, 1\}$ and χ be in $\{\alpha, \alpha^2\}$. These cases are:

- Case 1.a: $c_{i+1}c_i = \gamma 1$ or $c_{i+1}c_i = \gamma\alpha$, for all γ.
- Case 1.b: $c_{i+1}c_i = \gamma\alpha^2$, for all γ.
- Case 2: $c_{i+1}c_i = \chi 1$ or $c_{i+1}c_i = \chi\alpha$, for all χ.
- Case 3: $c_{i+1}c_i = \alpha\alpha^2$.
- Case 4: $c_{i+1}c_i = \alpha^2\alpha^2$.

The typical or default case is Case 1 (Case 1.a and Case 1.b combined).

The fourth and fifth steps are to find the encoding-decoding rule, which specifies the mapping from index to codeword and vice versa. This rule for $\mathcal{RC}_m^4$ is given in Theorem 4.

Theorem 4. *The relation between the lexicographic index* $g(\mathbf{c})$*,* $\mathbf{c} \in \mathcal{RC}_m^4$*, and the* 4*-ary codeword* $\mathbf{c}$ *itself is given by:*

$$g(\mathbf{c}) = \sum_{i=0}^{m-1} \Big[\big[(y_{i,1} + y'_{i,1}) a_i + y_{i,3} \big] N_4(i) + \big[2(y_{i,2} a_i + y_{i,3} - y'_{i,1}) + 5 y_{i,\mathrm{d}} \big] N_4(i-1)$$
$$\big[4(y'_{i,1} + y_{i,3}) + 2 y_{i,\mathrm{d}} \big] N_4(i-2) + 4 y_{i,\mathrm{d}} N_4(i-3) \Big], \tag{5.32}$$

where $y_{i,1}$*,* $y'_{i,1}$*,* $y_{i,2}$*,* $y_{i,3}$*, and* $y_{i,\mathrm{d}}$ *are specified as follows:*

$$\begin{aligned}
&y_{i,1} = 1 \textit{ if } c_{i+1}c_i \in \{\gamma 1, \gamma\alpha \mid \forall \gamma\}, \textit{ and } y_{i,1} = 0 \textit{ otherwise},\\
&y'_{i,1} = 1 \textit{ if } c_{i+1}c_i \in \{\gamma\alpha^2 \mid \forall \gamma\}, \textit{ and } y'_{i,1} = 0 \textit{ otherwise},\\
&y_{i,2} = 1 \textit{ if } c_{i+1}c_i \in \{\chi 1, \chi\alpha \mid \forall \chi\}, \textit{ and } y_{i,2} = 0 \textit{ otherwise},\\
&y_{i,3} = 1 \textit{ if } c_{i+1}c_i = \alpha\alpha^2, \textit{ and } y_{i,3} = 0 \textit{ otherwise},\\
&y_{i,\mathrm{d}} = 1 \textit{ if } c_{i+1}c_i = \alpha^2\alpha^2, \textit{ and } y_{i,\mathrm{d}} = 0 \textit{ otherwise}.
\end{aligned} \tag{5.33}$$

Proof. We compute the contributions $g_{i,j}(c_i)$ of a symbol c_i under Case j, for all j in $\{1, 2, 3, 4\}$, in a 4-ary LOCO codeword then merge them all. As for the typical case, Situation a, which we index by $1.a$, this contribution is the number of codewords starting with $c'_i < c_i$, where $c_i \in \{1, \alpha\}$, from the left in $\mathcal{RC}_{i+1}^4$. Thus using (5.24),

$$g_{i,1.a}(c_i) = a_i N_4(i + 1 - 1) = a_i N_4(i). \tag{5.34}$$

As for the typical case, Situation b, which we index by $1.b$, this contribution is the number of codewords starting with $c'_i < c_i$, where $c_i = \alpha^2$, from the left in $\mathcal{RC}_{i+1}^4$. Thus using (5.24), (5.25), and (5.26),

$$
\begin{aligned}
g_{i,1.b}(c_i) &= N_{4,1}(i+1) + N_{4,2}(i+1) + N_{4,3}(i+1) \\
&= 3N_4(i) - 2N_4(i-1) + 4N_4(i-2). \qquad (5.35)
\end{aligned}
$$

As for Case 2, this contribution is the number of codewords starting with $c_i'\gamma_1$, $c_i' < c_i$, where $c_i \in \{1, \alpha\}$ and $\gamma_1 \in \{0, 1\}$, from the left in $\mathcal{RC}_{i+1}^4$. Thus using (5.24),

$$
g_{i,2}(c_i) = a_i N_{4,1}(i) = 2a_i N_4(i-1). \qquad (5.36)
$$

As for Case 3, this contribution is the number of codewords starting with $\alpha c_i'$, $c_i' < c_i$, where $c_i = \alpha^2$, from the left in $\mathcal{RC}_{i+2}^4$. Those are all the codewords starting with $\gamma_1\gamma_2$, for all γ_1 and γ_2, from the left in $\mathcal{RC}_{i+1}^4$ plus all the codewords starting with α from the left in $\mathcal{RC}_{i+1}^4$. Thus using (5.24), (5.25), and (5.26),

$$
\begin{aligned}
g_{i,3}(c_i) &= 2N_{4,1}(i) + N_{4,2}(i+1) + N_{4,3}(i+1) \\
&= N_4(i) + 2N_4(i-1) + 4N_4(i-2). \qquad (5.37)
\end{aligned}
$$

As for Case 4, this contribution is the number of codewords starting with $\alpha^2 c_i'$, $c_i' < c_i$, where $c_i = \alpha^2$, from the left in $\mathcal{RC}_{i+2}^4$. Those are all the codewords starting with $\gamma_1\gamma_2$, for all γ_1 and γ_2, from the left in $\mathcal{RC}_{i+1}^4$ plus all the codewords starting with $\alpha^2\alpha$ from the left in $\mathcal{RC}_{i+2}^4$. Thus using (5.24), (5.28), and (5.29),

$$
\begin{aligned}
g_{i,4}(c_i) &= 2N_{4,1}(i) + N_{4,5}(i+2) + N_{4,6}(i+2) \\
&= 5N_4(i-1) + 2N_4(i-2) + 4N_4(i-3). \qquad (5.38)
\end{aligned}
$$

We use $y_{i,1}$, $y_{i,1}'$ (for Case 1), $y_{i,2}$ (for Case 2), $y_{i,3}$ (for Case 3), and $y_{i,\mathrm{d}}$ (for Case 4) from (5.33) along with a_i to merge (5.34), (5.35), (5.36), (5.37), and (5.38). We adopt the following merging functions, where $f^{\text{mer}}_{\ell}(\cdot)$ is associated with $N_4(i+1-\ell)$:

$$\begin{aligned}
f^{\text{mer}}_{1}(\cdot) &= (y_{i,1} + y'_{i,1})a_i + y_{i,3}, \\
f^{\text{mer}}_{2}(\cdot) &= 2(y_{i,2}a_i + y_{i,3} - y'_{i,1}) + 5y_{i,\text{d}}, \\
f^{\text{mer}}_{3}(\cdot) &= 4(y'_{i,1} + y_{i,3}) + 2y_{i,\text{d}}, \\
f^{\text{mer}}_{4}(\cdot) &= 4y_{i,\text{d}}.
\end{aligned} \tag{5.39}$$

Therefore, the general form of the symbol contribution $g_i(c_i)$ is:

$$g_i(c_i) = \sum_{\ell=1}^{4} f^{\text{mer}}_{\ell}(\cdot) N_4(i+1-\ell). \tag{5.40}$$

Substituting (5.39) and (5.40) in $g(\mathbf{c}) = \sum_{i=0}^{m-1} g_i(c_i)$ gives (5.32). ■

Remark 3. *Observe that the number of linearly independent merging variables is always less than the number of final cases [37]. Here, $y_{i,\text{d}}$ is dependent on the other merging variables as it can be written as $y_{i,\text{d}} = \mathbb{1}(a_i)(1 - y_{i,1} - y'_{i,1} - y_{i,2} - y_{i,3})$, where $\mathbb{1}(a_i) = 1$ if $a_i > 0$ and $\mathbb{1}(a_i) = 0$ if $a_i = 0$.*

For brevity, we again skip the sixth step, which is to assemble the encoding and decoding algorithms. These algorithms are a direct consequence of the rule in (5.32), and we refer the reader to [111], [35], [37], and [63] for details. Note that we sometimes refer to $\mathcal{RC}^4_m$ as a *1D* 4*-ary RR-LOCO code*.

5.5 Rate, Complexity, and Error Propagation

We start by calculating asymptotic rates. Unfortunately, deriving the capacity for 2D constrained codes is known to be notoriously hard. Therefore, we will derive the capacity $C^{\text{1D}}_{\mathcal{L}_q}$ only under the 1D constrained coding setup, which is already higher than the capacity under the 2D setup. Thus, $C^{\text{1D}}_{\mathcal{L}_q}$ serves as a ceiling for the highest achievable rate in a device where patterns

Table 5.1. Capacity Comparison Between $C^{\text{1D}}_{\mathcal{L}_q}$, 1D Binary RR Capacity $C^{\text{1D}}_{\text{RR2}}$, and 1D 4-ary RR Capacity $C^{\text{1D}}_{\text{RR4}}$

q	$C^{\text{1D}}_{\mathcal{L}_q}$	$C^{\text{1D}}_{\text{RR2}}$	Capacity gap $\%$	$C^{\text{1D}}_{\text{RR4}}$
4	0.8941	0.8471	5.257%	0.8859
8	0.9235	0.8981	2.750%	0.9239
16	0.9401	0.9235	1.766%	0.9429
32	0.9509	0.9388	1.272%	0.9544

in $\mathcal{L}_q$ are forbidden at least in one direction. We will shortly show that 1D constrained coding suffices in terms of performance.

An FSTD of a sequence where level patterns in $\mathcal{L}_q$ are forbidden is shown in Fig. 5.2. Based on this FSTD, the general adjacency matrix is (vectors are row vectors):

$$\mathbf{A}_1 = \left[\begin{array}{c|c|c|c}
\frac{q}{2} & \mathbf{1}_{\frac{q}{2}} & 0 & \mathbf{0}_{\frac{q}{2}-1} \\
\hline
\mathbf{0}^{\text{T}}_{\frac{q}{2}} & \mathbf{U}^{1}_{\frac{q}{2}} & \frac{q}{2}\mathbf{1}^{\text{T}}_{\frac{q}{2}} & \begin{array}{c} \mathbf{0}_{\frac{q}{2}-1} \\ \hline \mathbf{L}^{1}_{\frac{q}{2}-1} \end{array} \\
\hline
\frac{q}{2} & \mathbf{0}_{\frac{q}{2}} & 0 & \mathbf{0}_{\frac{q}{2}-1} \\
\hline
\mathbf{0}^{\text{T}}_{\frac{q}{2}-1} & \begin{array}{c|c} \mathbf{I}_{\frac{q}{2}-1} & \mathbf{0}^{\text{T}}_{\frac{q}{2}-1} \end{array} & \frac{q}{2}\mathbf{1}^{\text{T}}_{\frac{q}{2}-1} & \begin{array}{c} \mathbf{0}_{\frac{q}{2}-1} \\ \hline \begin{array}{c|c} \mathbf{L}^{1}_{\frac{q}{2}-2} & \mathbf{0}^{\text{T}}_{\frac{q}{2}-2} \end{array} \end{array}
\end{array}\right], \tag{5.41}$$

where $\mathbf{U}^{1}_{\delta}$ ($\mathbf{L}^{1}_{\delta}$) is an upper (lower) only-ones triangular matrix of size $\delta \times \delta$. Thus and from [103], the normalized capacity of a 1D constrained code forbidding the level patterns in $\mathcal{L}_q$ is:

$$C^{\text{1D}}_{\mathcal{L}_q} = \frac{\log_2(\lambda_{\max}(\mathbf{A}_1))}{\log_2 q}, \tag{5.42}$$

where $\lambda_{\max}(\mathbf{A})$ is the maximum real positive eigenvalue of the matrix $\mathbf{A}$.[5]

[5]For positive integers $a + b \leq q$, the set H of the a largest levels, and the set L of the b smallest levels in $\{0, 1, \dots, q-1\}$, a formula for the (count-constrained) capacity of the constrained system forbidding all level patterns in $\{\beta_1\beta_2\overline{\beta}_1 \mid \beta_1, \overline{\beta}_1 \in H, \beta_2 \in L\}$ was derived in [57].

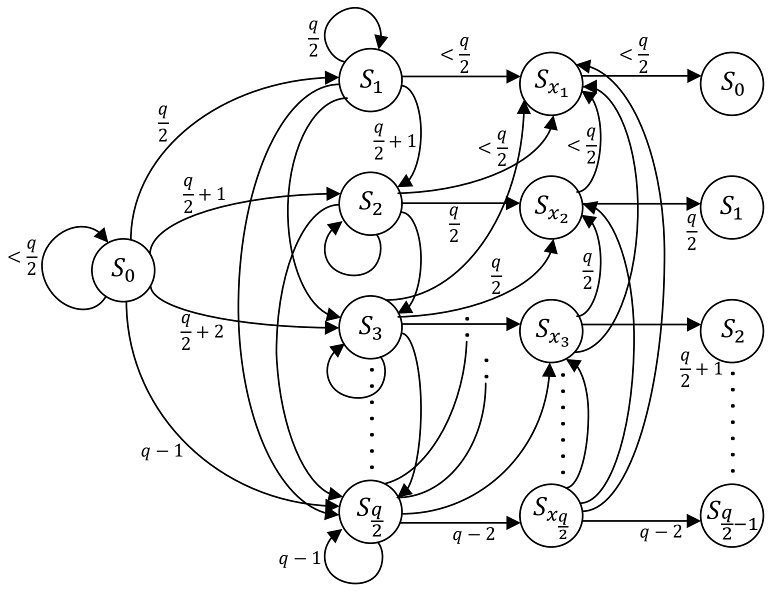

Figure 5.2. An FSTD of a 1D constrained sequence forbidding level patterns in $\mathcal{L}_q$, for any q. Here, we operate directly on level patterns for simplicity.

The capacity of a 2D binary code preventing $\{000, 010\}$ is the capacity of a 2D $(0, 1)$ RLL code, which is ≈ 0.5879 [58]. Thus, the normalized capacity of our 2D RR coding scheme is:

$$C^{\text{2D}}_{\text{RR2}} \approx \frac{0.5879 + \log_2 q - 1}{\log_2 q} = \frac{\log_2 q - 0.4121}{\log_2 q}. \tag{5.43}$$

As mentioned above, the 1D constrained system where patterns in $\mathcal{R}^2 = \{000, 010\}$ are forbidden can be interpreted as an interleaved RLL $(d, k) = (0, 1)$ constrained system, whose capacity is known to be $\log_2((1 + \sqrt{5})/2) \approx 0.6942$. Thus, the normalized capacity of our 1D RR-LOCO coding scheme is:

$$C^{\text{1D}}_{\text{RR2}} = \frac{\log_2((1 + \sqrt{5})/2) + \log_2 q - 1}{\log_2 q} \approx \frac{\log_2 q - 0.3058}{\log_2 q}. \tag{5.44}$$

The capacity gap between $C^{\text{1D}}_{\mathcal{L}_q}$ and $C^{\text{1D}}_{\text{RR2}}$ for different values of q is given in Table 5.1. The table shows that the capacity gap is small, and it gets even smaller as q increases.

The capacity $C^{\text{1D}}_{\mathcal{L}'_q}$ of a 1D constrained system where the level patterns in $\mathcal{L}'_q$ are forbidden is slightly higher than $C^{\text{1D}}_{\mathcal{L}_q}$ since $\mathcal{L}'_q \subset \mathcal{L}_q$. We skip the derivation of $C^{\text{1D}}_{\mathcal{L}'_q}$ for brevity.

An FSTD of a 1D 4-ary constrained system where patterns in $\mathcal{R}^4$ are forbidden is given in Fig. 5.3. The adjacency matrix is:

$$\mathbf{A}_2 = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 \end{bmatrix}.$$

The characteristic polynomial is:

$$\det(x\mathbf{I} - \mathbf{A}_2) = x^6 - 3x^5 + 2x^4 - 9x^3 - 7x^2 - 6x - 4. \tag{5.45}$$

We can see that if x is replaced by $\lambda_{\text{c}} = \lambda_{\max}(\mathbf{A}_2)$, we get:

$$\lambda_{\text{c}}^m = 3\lambda_{\text{c}}^{m-1} - 2\lambda_{\text{c}}^{m-2} + 9\lambda_{\text{c}}^{m-3} + 7\lambda_{\text{c}}^{m-4} + 6\lambda_{\text{c}}^{m-5} + 4\lambda_{\text{c}}^{m-6}, \tag{5.46}$$

which is consistent with the cardinality recursion in (5.22). The capacity of this 4-ary constrained system is $\log_2(\lambda_{\max}(\mathbf{A}_2)) = \log_2(3.4147) = 1.7718$ bits/symbol. Thus, the normalized capacity of our 1D 4-ary RR-LOCO coding scheme is:

$$C^{\text{1D}}_{\text{RR4}} = \frac{1.7718 + \log_2 q - 2}{\log_2 q} \approx \frac{\log_2 q - 0.2282}{\log_2 q}. \tag{5.47}$$

Table 5.1 shows the capacity gain achieved by the 1D 4-ary RR scheme over the 1D binary RR schemes, and we will show that the performance, i.e., the Flash device protection, is nearly the same. An interesting observation is that for $q \in \{8, 16, 32\}$, the capacity of our 1D

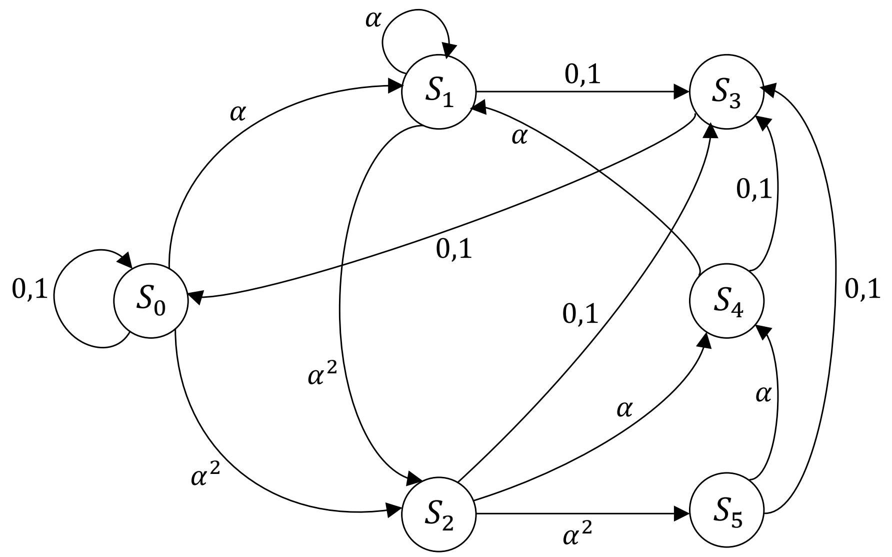

Figure 5.3. An FSTD of a 1D 4-ary constrained sequence forbidding patterns in $\mathcal{R}^4$.

4-ary RR scheme $C^{\text{1D}}_{\text{RR4}}$ is slightly higher than $C^{\text{1D}}_{\mathcal{L}_q}$.

Next, we discuss the finite-length rates. First, the normalized rate of our 2D binary RR constrained coding scheme is:

$$R^{\text{2D}}_{\text{RR2}} = \frac{0.5 + \log_2 q - 1}{\log_2 q} = \frac{\log_2 - 0.5}{\log_2 q} \tag{5.48}$$

since the rate of our left-most page coding is 0.5.

Regarding our 1D binary RR-LOCO coding scheme, we bridge with the pattern 11 between consecutive codewords in $\mathcal{RC}^2_m$ on the left-most page, and we remove the codeword $\mathbf{1}^m$ for self-clocking [35, 37]. Thus, the rate on the left-most page is $\lfloor \log_2(N_2(m) - 1) \rfloor/(m + 2)$, and the normalized rate of our 1D binary RR-LOCO coding scheme is:

$$R^{\text{1D}}_{\text{RR2}} = \frac{1}{\log_2 q}\left[\frac{\lfloor \log_2(N_2(m) - 1) \rfloor}{m + 2} + \log_2 q - 1\right]. \tag{5.49}$$

1D binary RR-LOCO coding schemes are capacity-achieving schemes in the sense that the limit as $m \rightarrow \infty$ of $R^{\text{1D}}_{\text{RR2}}$ is $C^{\text{1D}}_{\text{RR2}}$ (see also [35]). Another capacity-achieving 1D RR constrained coding scheme, implementable using enumerative coding without the need for bridging bits, can be obtained by interleaving codewords from an optimal block code for the RLL $(d, k) = (0, 1)$ constraint [81] on the left-most pages. LOCO codes, however, offer simplicity and reconfigurability, which is important as the device ages [35].

Regarding our 1D 4-ary RR-LOCO coding scheme, we cannot bridge with a single GF(4) symbol between consecutive codewords in $\mathcal{RC}_m^4$ on the two left-most pages since any symbol separating α^2 and α^2 generates a forbidden pattern. We propose a novel two-symbol bridging in which *we can encode input information bits within the bridging interval* as follows:

- For input information bits $00 \in$ GF(2), bridge with $00 \in$ GF(4).
- For input information bits $01 \in$ GF(2), bridge with $01 \in$ GF(4).
- For input information bits $10 \in$ GF(2), bridge with $10 \in$ GF(4).
- For input information bits $11 \in$ GF(2), bridge with $11 \in$ GF(4).

While it has no effect on the asymptotic rate, this bridging scheme remarkably reduces the code length at which a specific rate is achieved, significantly reducing the complexity and error propagation in consequence.

To achieve self-clocking, we remove the two codewords $\mathbf{0}^m$ and $\mathbf{1}^m$, which is expected given the bridging above [35, 37]. Thus, the rate on the two left-most pages is $(\lfloor \log_2(N_4(m) - 2)\rfloor + 2)/(m + 2)$ bits/symbol, and the normalized rate of our 1D 4-ary RR-LOCO coding scheme is:

$$R_{\text{RR4}}^{\text{1D}} = \frac{1}{\log_2 q}\left[\frac{\lfloor \log_2(N_4(m) - 2)\rfloor + 2}{m + 2} + \log_2 q - 2\right]. \qquad (5.50)$$

1D 4-ary RR-LOCO coding schemes are capacity-achieving schemes in the sense that the limit as $m \to \infty$ of $R_{\text{RR4}}^{\text{1D}}$ is $C_{\text{RR4}}^{\text{1D}}$. RR-LOCO codes offer simplicity and reconfigurability, which is important as the device ages [35].

The 2D binary RR constrained coding scheme we propose requires no additional complexity for encoding and decoding since data is written/read directly to/from specific positions on the left-most page and directly to/from all positions on other pages. As for the 1D binary RR-LOCO coding scheme, the complexity is governed by the size of the adder that executes the encoding-decoding rule, which is:

$$s_2 = \lfloor \log_2(N_2(m) - 1) \rfloor \tag{5.51}$$

bits. Similarly and as for the 1D 4-ary RR-LOCO coding scheme, the complexity is governed by the adder size, which is:

$$s_4 = \lfloor \log_2(N_4(m) - 2) \rfloor \tag{5.52}$$

bits. For ease of implementation and to avoid affecting the access speed, we prefer to apply the 1D RR-LOCO coding schemes along wordlines instead of bitlines since the performance is very close, as demonstrated by the experimental results in Section 5.6.

Error propagation is the phenomenon that a single writing error results in multiple errors while reading. The 2D binary RR coding scheme does not incur any error propagation. Thus, the error propagation factor of it is $E^{\text{2D}}_{\text{RR2}} = 1$. As for the 1D binary RR-LOCO coding scheme, there is no codeword-to-codeword error propagation. However, there exists limited error propagation resulting from the codeword-to-message conversion [34, 35] on the left-most page only. This error propagation reaches $s_2/2$ bits on average, where s_2 is the message length as well from (5.51). Consequently, the error propagation factor averaged over $\log_2 q$ pages is:

$$E^{\text{1D}}_{\text{RR2}} = \frac{1}{\log_2 q}\left[\frac{s_2}{2} + \log_2 q - 1\right]. \tag{5.53}$$

As for the 1D 4-ary RR-LOCO coding scheme, again there exists limited error propagation resulting solely from the LOCO codeword-to-message conversion [34, 35] on the two left-most pages. This error propagation reaches $s_4/2$ bits on average, where s_4 is the message length as well from (5.52). Observe that there is no error propagation for the two additional bits encoded at each bridging interval to specify the two 4-ary bridging symbols. Therefore, the

Table 5.2. Comparisons of Rate, Complexity, and Error Propagation at the Same Length Between 2D RR and 1D Binary RR Constrained Coding Schemes

q	m	$R^{\text{2D}}_{\text{RR2}}$	$R^{\text{1D}}_{\text{RR2}}$	s_2	$E^{\text{2D}}_{\text{RR2}}$	$E^{\text{1D}}_{\text{RR2}}$
4	7	0.7500	0.7778	5	1.000	1.750
4	11	0.7500	0.8077	8	1.000	2.500
4	21	0.7500	0.8261	15	1.000	4.250
8	7	0.8333	0.8519	5	1.000	1.500
8	11	0.8333	0.8718	8	1.000	2.000
8	21	0.8333	0.8841	15	1.000	3.167
16	7	0.8750	0.8889	5	1.000	1.375
16	11	0.8750	0.9038	8	1.000	1.750
16	21	0.8750	0.9130	15	1.000	2.625

average error propagation on any of these two left-most pages is:

$$\frac{s_4}{2} \cdot \frac{m}{m+2} + 1 \cdot \frac{2}{m+2} = \frac{s_4 m + 4}{2(m+2)}. \tag{5.54}$$

Consequently, the error propagation factor averaged over $\log_2 q$ pages is:

$$E^{\text{1D}}_{\text{RR4}} = \frac{1}{\log_2 q}\left[2 \cdot \frac{s_4 m + 4}{2(m+2)} + \log_2 q - 2\right] = \frac{1}{\log_2 q}\left[\frac{s_4 m + 4}{m+2} + \log_2 q - 2\right]. \tag{5.55}$$

Another metric to compare 1D binary with 1D 4-ary RR-LOCO coding schemes is the amount of coded data at a given rate. As this amount decreases, the code allows achieving the desired rate at a smaller length m, which is an advantage. Since for our 1D binary and 1D 4-ary RR-LOCO coding schemes we use two bits and two symbols for bridging, respectively, these amounts of coded data, $D^{\text{1D}}_{\text{RR2}}$ (binary) and $D^{\text{1D}}_{\text{RR4}}$ (4-ary) are:

$$D^{\text{1D}}_{\text{RR2}} = (m+2)\log_2 q,\ m \text{ is the length of } \mathcal{RC}^2_m, \tag{5.56}$$

$$D^{\text{1D}}_{\text{RR4}} = (m+2)\log_2 q,\ m \text{ is the length of } \mathcal{RC}^4_m. \tag{5.57}$$

Table 5.2 gives the normalized rates, adder sizes, and error propagation factors of the proposed binary RR schemes under various parameters. The 1D binary RR-LOCO coding

Table 5.3. Comparisons of Minimum Coded Data, Complexity, and Error Propagation to Achieve Certain Rate Between 1D Binary RR and 1D 4-ary RR Constrained Coding Schemes

q	Rate	$D^{\text{1D}}_{\text{RR2}}$	$D^{\text{1D}}_{\text{RR4}}$	s_2	s_4	$E^{\text{1D}}_{\text{RR2}}$	$E^{\text{1D}}_{\text{RR4}}$
8	0.8500	27	21	5	9	1.500	2.667
8	0.8750	48	24	10	11	2.333	3.250
8	0.8900	138	48	31	25	5.833	7.708
8	0.9000	–	60	–	32	–	10.000
16	0.8900	48	28	7	9	1.625	2.250
16	0.9050	64	32	10	11	2.000	2.688
16	0.9150	144	48	24	18	3.750	4.333
16	0.9200	288	64	49	25	6.875	6.031
16	0.9300	–	100	–	41	–	9.970

scheme has a remarkable rate advantage that reaches 10.147%, 6.096%, and 4.343% for $q = 4$, $q = 8$, and $q = 16$, respectively, over the 2D binary RR constrained coding scheme. The 2D binary RR scheme has a clear advantage in terms of both complexity and error propagation as it requires no processing to encode and decode. Having said that, the error propagation factor of the 1D binary RR scheme decreases notably as q increases. For example, $E^{\text{1D}}_{\text{RR2}} = 2.625$ for $q = 16$ and $m = 21$, which is remarkably small given the code length.

In Table 5.3, we compare 1D binary with 1D 4-ary RR-LOCO coding schemes in a different way. In particular, we fix the normalized rate, and find the minimum amount of coded data and the minimum complexity (adder size) required to achieve this desired rate for the two coding schemes, in addition to the minimum error propagation associated with them.[6] The sign "–" is used in the table whenever the binary coding scheme cannot achieve such a rate. The main conclusions from Table 5.3 are:

- For $q = 8$ and $q = 16$, the 4-ary coding scheme requires less coded data (smaller lengths) than the binary coding scheme does for all desired rates. The difference in favor of the 4-ary coding scheme increases as the rate increases.
- At lower rates, the complexity of the binary coding scheme is lower than that of the 4-ary

[6]Achieving a desired rate here means reaching a normalized rate greater than or equal to this desired rate.

coding scheme. However, at rates ≥ 0.8900 for $q = 8$ and ≥ 0.9150 for $q = 16$, the 4-ary coding scheme wins the complexity competition.

- As expected, the binary coding scheme incurs less error propagation in general because LOCO coding is performed on one page only. However, at higher rates and higher q, the 4-ary coding scheme becomes quite competitive to the intriguing extent that it already incurs less error propagation at rate 0.9200 and $q = 16$.

The 1D and 2D RR coding schemes can be used in the same device, but at different lifetime stages. A 1D RR-LOCO coding scheme, binary or 4-ary, can be used when the device is relatively fresh or until a moderate number of program/erase (P/E) cycles, while the 2D RR constrained coding scheme can be used when the device ages, where preventing the error-prone patterns in both directions could make a difference and the associated rate loss could be acceptable. However, this performance difference is shown to be small in Section 5.6, at least for the TLC Flash device we used. The section also shows that the performance difference between 1D binary and 1D 4-ary RR-LOCO coding schemes is negligible.

Remark 4. *An idea that allows page separation for MLC Flash was introduced in [112]. However, the rate offered is only* 0.7500*, which is significantly below the rates offered via our 1D binary RR coding scheme for MLC. Another idea that allows page separation for TLC Flash was introduced in [88]. However, it only heuristically addresses the level pattern* 707.

5.6 Experimental Results on TLC Flash

To characterize the performance of the proposed RR constrained coding schemes, we conducted program/erase (P/E) cycling experiments on several blocks of a commercial 1X-nm TLC Flash chip, as follows:

1. Erase Flash memory block under test.

2. Program all pages of block under test with data. For uncoded experiments, program pseudorandom data at each P/E cycle. For RR experiments, program prepared data satisfying RR constraints at each P/E cycle.

3. For each successive P/E cycle of RR experiments, "rotate" the data, so the data that was written on the page i is written on the page $(i + 1)$, wrapping around the last page to the first page.

4. Record bit errors and compute channel bit error rate (BER) every 100 P/E cycles.

The PE cycling experiments were performed at room temperature in a continuous manner with no wait time between the erase-program-read operations.

Gray mappings used in Flash devices may vary between manufacturers and product generations. In our preliminary work [39], we modified the forbidden binary patterns in accordance with the device mapping so that RR coding on one page per wordline would eliminate most of the patterns in $\mathcal{L}_q$ that induce the most severe ICI (see Remark 2).

In this work, the 8-ary encoded level sequences generated by the RR encoders described herein using the RAGM mapping were translated according to the device Gray mapping into the corresponding binary sequences for the lower, middle, and upper pages in the TLC Flash memory. Thus, the 8-ary level sequences stored in the memory are precisely the RR-encoded level sequences (each cell is programmed to a level in $\{0, 1, \ldots, q - 1\}$).

The left subfigure in Fig. 5.4 shows the channel BER from P/E cycle 0 to P/E cycle $10{,}000$ using uncoded pseudorandom data, a rate 24:36 1D binary RR-LOCO code along wordlines or bitlines, and a rate 20:12 bits/symbol 1D 4-ary RR-LOCO code along wordlines or bitlines. The right subfigure in Fig. 5.4 shows the channel BER from P/E cycle $4{,}000$ to P/E cycle $10{,}000$ for these cases in more detail. Note that the binary RR code and 4-ary RR code have the same overall rate: $R^{\text{1D}}_{\text{RR2}} = 8/9 \approx 0.8889$ using (5.49) and $R^{\text{1D}}_{\text{RR4}} = 8/9 \approx 0.8889$ using (5.50). Therefore, the 1D binary coding scheme achieves about 99% (96%) of the capacity $C^{\text{1D}}_{\text{RR2}}$ ($C^{\text{1D}}_{\mathcal{L}_q}$) and the 1D 4-ary coding scheme achieves about 96% of the capacity $C^{\text{1D}}_{\text{RR4}}$.

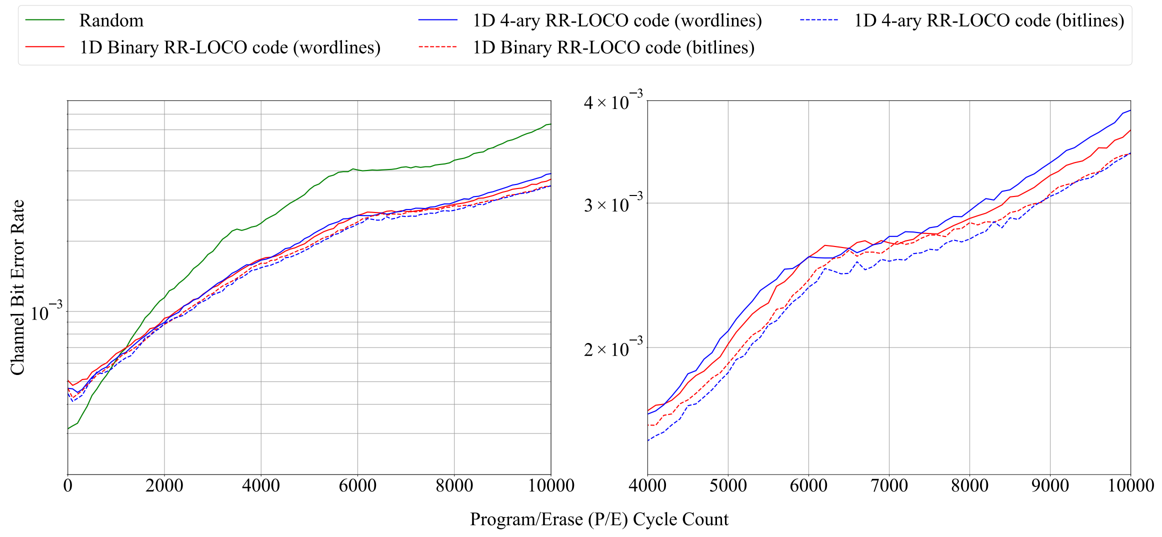

Figure 5.4. (Left) Measured average channel BER comparison when all pages are programmed with random data (green curve), 1D binary RR-LOCO coded data (red curves) along wordlines (solid curve) or bitlines (dashed curve), and 1D 4-ary RR-LOCO coded data (blue curves) along wordlines (solid curve) or bitlines (dashed curve) from P/E cycle 0 to P/E cycle $10{,}000$. (Right) Measured average channel BER excluding random data from P/E cycle $4{,}000$ to P/E cycle $10{,}000$.

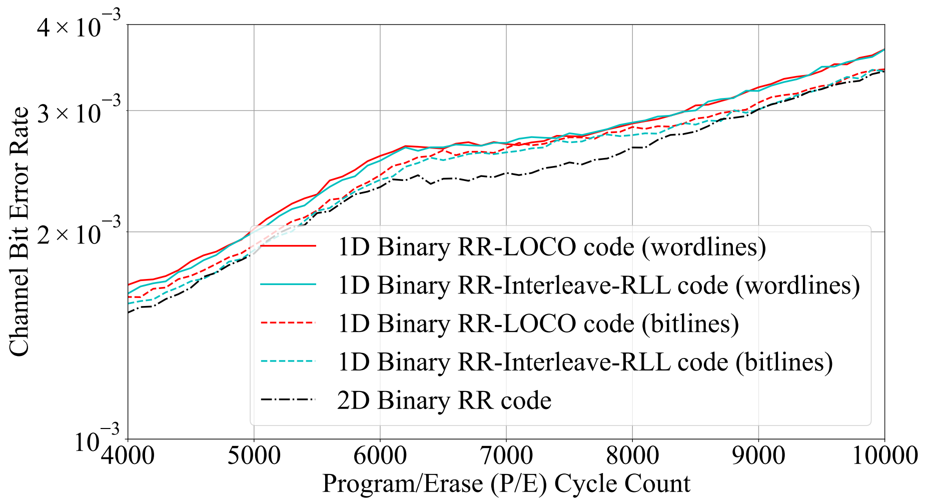

Figure 5.5. Measured average channel BER comparison of 1D binary RR-LOCO coded data (red curves) along wordlines (solid curve) or bitlines (dashed curve), 1D binary interleaved RLL-$(0, 1)$ coded data (cyan curves) along wordlines (solid curve) or bitlines (dashed curve), and 2D binary RR coded data (black curve) from P/E cycle $4{,}000$ to P/E cycle $10{,}000$.

As shown in Fig. 5.4, the uncoded performance is better than that of both binary and 4-ary RR codes up to around $1{,}200$ P/E cycles and becomes notably worse thereafter. At the later stages of P/E cycling, ICI becomes severe and the RR codes achieve significantly lower channel BER than uncoded data. Specifically, 1D binary RR-LOCO codes along wordlines increase device lifetime by about $1{,}800$ P/E cycles when channel BER is 2×10^{-3}, representing a 57% lifetime gain, and achieve about $3{,}700$ P/E cycles gain when channel BER is 3×10^{-3}, corresponding to a 79% lifetime gain. As shown in the right subfigure of Fig. 5.4, the BER of the 1D binary RR code along wordlines is almost the same as that of the 4-ary RR code between $2{,}000$ and $8{,}000$ P/E cycles. When the P/E cycle count is larger than $8{,}000$, the BER of the 1D binary RR code along wordlines is slightly better than that of the 1D 4-ary RR code. In particular, when channel BER is 3×10^{-3}, the 1D binary RR code along wordlines provides a lifetime that is about 300 P/E cycles larger that than obtained with the 1D 4-ary RR code along wordlines. Along the bitline direction, quite intriguingly, the performance of the 1D 4-ary RR-LOCO code is generally better than, though close to, that of the 1D binary RR-LOCO code. The advantage of the 1D 4-ary RR-LOCO is most pronounced from P/E cycle $6{,}300$ to P/E cycle $8{,}300$. For both binary and 4-ary coding, coding along bitlines generally offers better performance than coding along wordlines.

Fig. 5.5 compares the BER performance of different implementations of binary RR codes at high P/E cycles: the 24:36 1D binary RR-LOCO code along the wordline or bitline direction, the 1D binary interleaved 12:18 RLL $(d, k) = (0, 1)$ code (which has an overall block length 36 after interleaving) along the wordline or bitline direction, and the 2D binary RR code. Using (5.48), we obtain $R^{\text{2D}}_{\text{RR2}} = 5/6 \approx 0.8333$. Therefore, the 2D coding scheme achieves about 93% (90%) of the capacity $C^{\text{2D}}_{\text{RR2}}$ ($C^{\text{1D}}_{\mathcal{L}_q}$).

Referring to Fig. 5.5, we make the following observations at all P/E cycles: each 1D RR coding scheme along the bitline direction achieves a slightly better channel BER performance than along the wordline direction; the 1D RR coding schemes along the same direction have similar performance; and the performance of the 2D RR constrained code is better than that of the 1D RR codes along any one direction. As a result, when channel BER is 2×10^{-3}, the 2D binary RR coding increases lifetime by 100 P/E cycles over the 1D binary RR-LOCO coding along bitlines and 300 P/E cycles over the 1D binary RR-LOCO coding along wordlines. When channel BER is 3×10^{-3} and the wear condition of the Flash device is more severe, the 2D binary RR coding outperforms the 1D binary RR-LOCO coding along bitlines by about 200 P/E cycles and the 1D binary RR-LOCO coding along wordlines by about 600 P/E cycles.

These measurements confirm some of the claimed practical advantages of 4-ary RR codes. The performance results of the 1D binary RR-LOCO code and the 1D 4-ary RR-LOCO code along both wordline and bitline directions are very similar, and the designed codes have the same overall rate (including bridging symbols). The 1D 4-ary RR-LOCO code has a shorter overall block length corresponding to 12 bits per coded page (10 symbols plus 2 bridging symbols) in comparison to the 1D binary RR-LOCO code which has overall block length of 36 bits on the coded page. Moreover, in the code design, the 1D 4-ary RR-LOCO encoder uses an adder size of 18 bits, while the 1D binary RR-LOCO requires an adder size of 24 bits.

An examination of level probabilities induced by 1D binary and 1D 4-ary RR constraints provides some intuitive insight into the experimental results in Figs. 5.4 and 5.5. The probabilities of binary symbols 0 and 1 under the RLL $(d, k) = (0, 1)$ constraint are approximately 0.2764 and 0.7236, respectively [107]. Asymptotically, this leads to probabilities of individual symbols corresponding to levels in $\mathcal{V}_0 = \{4, 5, 6, 7\}$ and $\mathcal{V}_1 = \{0, 1, 2, 3\}$ of about 0.0691 and 0.1809, respectively. From the FSTD of the 4-ary constraint forbidding patterns in $\mathcal{R}_4$, shown in Fig. 5.3, we find that the probabilities of individual symbols corresponding to levels in $\mathcal{W}_0 = \{6, 7\}$, $\mathcal{W}_1 = \{4, 5\}$, $\mathcal{W}_2 = \{2, 3\}$, and $\mathcal{W}_3 = \{0, 1\}$ are about 0.0787, 0.1030, 0.1591, and 0.1591, respectively. Bridging symbols change these probabilities slightly, further increasing the probabilities of symbols corresponding to levels in $\{0, 1, 2, 3\}$ relative to symbols corresponding to levels in $\{4, 5, 6, 7\}$. These probabilities contrast with those of uncoded random data, where each symbol/level has the same probability of $1/8 = 0.125$.

The modified symbol probabilities help to explain the relative performances of the

1D binary RR codes in the wordline and bitline directions as well as their performance relative to the 2D RR code. Applying 1D binary RR coding in the wordline direction indirectly reduces the probability of detrimental patterns in the bitline direction, and vice versa. This reduces the expected advantage of bitline coding over wordline coding resulting from more severe ICI in the bitline direction. For similar reasons, the advantage of 2D coding over 1D coding in either direction is less than expected (even without taking into account the additional rate penalty associated with 2D coding).

We remark that the designed codes are efficient, with rates fairly close to capacity, and the symbol and pattern probabilities observed in the data written to the Flash memory are close to the theoretical values mentioned above.

The cross-over behavior observed in Fig. 5.4 can be explained if the level patterns eliminated by the code, especially ICI-prone patterns, are not the only significant contributors to error early in the device lifetime. The binary RR coding significantly changes level probabilities compared with the uncoded setting, possibly increasing the probability of some of the remaining level patterns that cause errors due to other effects, and accordingly increasing their contribution to the BER at low P/E cycles. One way to address this issue is to delay the introduction of coding until later P/E cycles when ICI affects performance. Alternatively, one might apply different constraints at different P/E cycles before and after the cross-over point, much as adaptive error correction code designs have been proposed to achieve different degrees of protection at various stages of the flash device lifetime [16, 69]. The reconfigurability feature of LOCO code designs could be exploited, and a machine learning module could be used to identify the device status and direct the transition from one code to another at the appropriate time based on that status. In this regard, we also note that machine learning modeling, as proposed in [130], can be used to characterize the spatio-temporal ICI effects of the Flash memory device and provide a tool for optimizing the offline design of RR-LOCO codes. Another possible approach is to optimize system performance through a combination of signal processing methods [25], [3] and RR coding. These ideas represent directions for future research.

5.7 Conclusion

We introduced read-and-run (RR) constrained coding schemes for modern Flash devices. RR coding schemes eliminate patterns prone to ICI-induced errors while allowing systematic encoder and decoder implementations, high overall rates, and page separation in data recovery. We analyzed properties of 1D binary RR-LOCO codes, 1D 4-ary RR-LOCO codes, and a 2D binary RR code. The three RR coding schemes offer different advantages, and we suggest that system requirements at different stages of the device lifetime should determine the most suitable scheme or schemes to use. Experimental results reveal significant P/E-cycle lifetime gains in a commercial Flash device. Future work includes the incorporation of LDPC codes [36] with RR coding schemes and the development of machine learning-aided, reconfigurable RR coding schemes to maximize Flash device lifetime.

Acknowledgement

This chapter is in part a reprint of the material in the paper [38]: Ahmed Hareedy, Simeng Zheng, Paul H. Siegel, and Robert Calderbank, "Efficient constrained codes that enable page separation in modern flash memories," *IEEE Transactions on Communications*, vol. 71, no. 12, pp. 6834–6848, Dec. 2023. The work was also presented in part at ICC 2022 [39] and NVMW 2024 [40]: A. Hareedy, S. Zheng, P. H. Siegel, and R. Calderbank, "Read-and-run constrained coding for modern flash devices," in *Proc. IEEE International Conference Communications (ICC)*, Seoul, South Korea, May 2022; Ahmed Hareedy, Simeng Zheng, Paul H. Siegel, and Robert Calderbank, "Read-and-run constrained coding for modern flash memories," in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024. The dissertation author would like to thank the collaborators in these publications.

Chapter 6

Code-Aware Storage Channel Modeling via Machine Learning

6.1 Introduction

Recently, there has been great interest in the application of machine learning in communications and networking, including data storage. For example, robust signal detection in magnetic recording channels using a recurrent neural network (RNN) architecture was demonstrated in [128]. A low-density parity-check (LDPC) decoder with flexible code lengths and column weights exploiting RNN was proposed in [123]. Machine learning was also applied to page failure prediction [76, 109] and, in a limited setting, read voltage generation [77] in NAND flash memory. The synergy between machine learning and data storage is stimulating important mutual progress.

Realistic models for storage and communication channels are critical tools in the design of signal processing and coding methods. Generative flash modeling (GFM) [127] was recently proposed to model the complex spatio-temporal characteristics of read voltages in flash memory channels. Although statistical models [60, 67, 78, 95] to characterize the effects of P/E cycling, ICI, and retention on flash memory read voltages have been proposed, their predictions have not been validated in the literature by comparing to measurements of pattern-dependent errors as a function of spatial and temporal factors.

GFM uses a conditional VAE-GAN [64] architecture, combining a variational autoencoder (VAE) [62] and a generative adversarial network (GAN) [31] in a conditional setting. This modeling approach was shown to comprehensively learn both spatial and temporal properties of the flash channel.

Constrained codes [80] have been proposed to mitigate read errors arising from the ICI phenomenon in flash memory by forbidding the programming of error-prone patterns. Learning the characteristics of the input-constrained flash channel poses a challenge, however. Statistical models have not been used to explore the subtle characteristics of the channel associated with the use of constrained data. GFM has the potential to model the input-constrained channel, but a model trained from pseudo-random data does not provide sufficient knowledge about the constrained channel. On the other hand, acquiring a large dataset of measurements from constrained data can consume excessive amounts of time and hardware resources.

It has been observed that learned knowledge from models pre-trained on a large dataset (e.g., ImageNet [24]) can effectively be applied to other tasks, either by extracting off-the-shelf features from trained networks [98,124], or by adapting learned knowledge to a new domain [92]. Moreover, transferring learning has shown success in the context of generative models, e.g., in applications to image generation [120]. To accurately model the input-constrained flash channel, we therefore propose a transfer learning approach, whereby the GFM network is first trained on a large dataset of measurements from pseudo-random data, and then is fine-tuned by re-training on a much smaller dataset of measurements from constrained data. We refer to this as code-aware GFM.

The chapter has the following contributions:

1. We propose a novel framework for code-aware generative channel modeling, where the voltage levels of coded program levels can be precisely and rapidly reconstructed.

2. We show how generative models trained on pseudo-random programming data can efficiently transfer knowledge to other coded-channel modeling tasks where code-specific data is limited.

3. We demonstrate the quality of reconstruction in code-aware GFM by analysis of voltage distributions and bit error rates (BERs).

6.2 NAND Flash Memory and ICI Mitigation

6.2.1 NAND Flash Memory Basics

NAND flash memory stores data as voltages in floating gate transistors, called cells. In a flash chip, cells are organized in two-dimensional (2-D) arrays, called blocks, consisting of horizontal wordlines (WLs) and vertical bitlines (BLs). Multilevel flash memories store multiple

Table 6.1. Numerical Values of Pattern-Dependent Error Rates for the Most Severe ICI Patterns

	Error rate	$\begin{bmatrix} 7 & 0 & 7 \end{bmatrix}$	$\begin{bmatrix} 7 & 0 & 6 \end{bmatrix}$	$\begin{bmatrix} 6 & 0 & 7 \end{bmatrix}$	$\begin{bmatrix} 7 \\ 0 \\ 7 \end{bmatrix}$	$\begin{bmatrix} 7 \\ 0 \\ 6 \end{bmatrix}$	$\begin{bmatrix} 6 \\ 0 \\ 7 \end{bmatrix}$	$\begin{bmatrix} & 7 & \\ 7 & 0 & 7 \\ & 7 & \end{bmatrix}$	$\begin{bmatrix} & 7 & \\ 7 & 0 & 6 \\ & 7 & \end{bmatrix}$	$\begin{bmatrix} & 7 & \\ 7 & 0 & 7 \\ & 6 & \end{bmatrix}$
4000 P/E	2.45%	10.97%	7.42%	7.36%	15.42%	10.76%	8.78%	48.06%	35.64%	36.59%
7000 P/E	4.06%	14.45%	10.35%	10.15%	20.48%	15.01%	12.81%	50.34%	42.15%	37.39%
10000 P/E	5.84%	18.42%	13.31%	13.46%	25.73%	19.35%	17.11%	54.22%	44.42%	44.67%

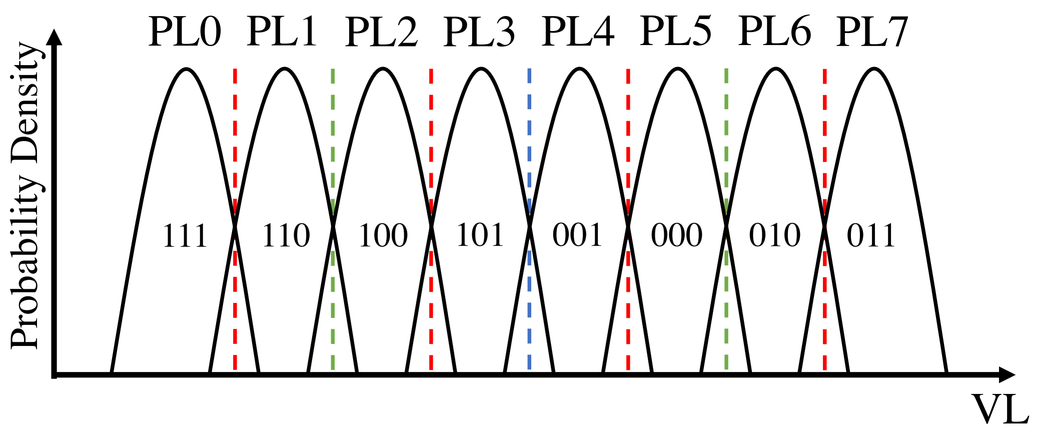

Figure 6.1. Voltage distributions and a recursive alternate Gray mapping (RAGM) between cell program levels and binary logic values of a TLC NAND flash memory.

bits per cell. For example, a triple-level cell (TLC) memory stores three bits using $2^3=8$ possible voltage levels. Within each block, the three bits stored in cells along a WL are logically grouped into three pages, called the lower, middle, and upper page, respectively.

There are three basic operations on a flash device: program (write), read, and erase. We denote the program level as PL and the read voltage level as VL. Fig. 6.1 illustrates the the conditional probability density functions (PDFs) of read voltages for 8 program levels, each corresponding to a 3-bit string of lower, middle, and upper bits. The dash-dotted vertical lines represent the read thresholds used to recover the stored data. Level errors and bit errors occur when, for example, PL=0 induces a read voltage VL lying above the first threshold and below the second threshold, causing the level to be mistakenly detected as 1 and the the upper bit to be mistakenly detected as 0.

6.2.2 ICI Mitigation via Constrained Coding

ICI effects, caused by parasitic capacitive coupling between flash cells, is one of the major obstacles to accurate programming and reading of a flash device [10]. Severe ICI arises when three consecutive cells in WL or BL directions are programmed to high-low-high levels.

Error rates for the most severe ICI patterns in a commercial TLC flash device are shown in Table 6.1. Using (i, j) to denote the (WL,BL) position of a cell in the block and $V_{th(01)}$ to denote the threshold between PL$=0$ and PL$=1$, the table gives the overall level error rate for PL$=0$, and the error rates for worst-case WL, BL, and 2-D patterns, or, mathematically,

$$\begin{aligned}
&P(\mathrm{VL}_{(i,j)} > V_{th(01)}|\mathrm{PL}_{(i,j)}{=}0);\\
&P(\mathrm{VL}_{(i,j)} > V_{th(01)}|\mathrm{PL}_{(i,j-1)}, \mathrm{PL}_{(i,j)}{=}0, \mathrm{PL}_{(i,j+1)});\\
&P(\mathrm{VL}_{(i,j)} > V_{th(01)}|\mathrm{PL}_{(i-1,j)}, \mathrm{PL}_{(i,j)}{=}0, \mathrm{PL}_{(i+1,j)});\\
&P(\mathrm{VL}_{(i,j)} > V_{th(01)}|\mathrm{PL}_{(i\pm1,j)}, \mathrm{PL}_{(i,j)}{=}0, \mathrm{PL}_{(i,j\pm1)}).
\end{aligned} \tag{6.1}$$

We make two observations from Table 6.1. First, ICI significantly increases error rates. At 4000 P/E cycles, the error rate of 707 pattern in WL (resp., BL) direction is a factor of 4.5 (resp., 6.3) larger than the average error rate. If we program 707 in both directions, the error rate is a factor of 19.6 larger than the average error rate. Second, P/E cycling causes error rates to increase. Specifically, the average error rate increases by a factor of 2.38 from 4000 P/E cycles to 10000 P/E cycles. For dominant 707 error patterns in WLs (resp., BLs), the error rate increases by a factor of 1.68 (resp., 1.67) from 4000 P/E cycles to 10000 P/E cycles.

Solid-state drives (SSDs) employ powerful error-correction codes (ECCs) [46] within their controllers to cope with such errors. Constrained codes to further reduce ICI-induced errors have been proposed and some have been experimentally validated [35, 39, 97, 112]. In particular, read-and-run (RR) constrained coding techniques [39] efficiently eliminate selected detrimental patterns by coding on only one page per WL. They allow random page access and are compatible with page-based ECCs. A generative model that accurately learns input-constrained channels will be a valuable tool in optimizing the combination of constrained coding and ECC.

6.3 Code-aware Storage Channel Modeling

The GFM scheme in [127] learns an approximation to the intractable likelihood $P(\text{VL}|\text{PL}, \text{P/E})$ from a dataset of measured voltage arrays VL produced by pseudo-random (unconstrained) program arrays PL. The goal of code-aware channel modeling is to infer the intractable likelihood $P(\text{VL}^{\mathcal{S}}|\text{PL}^{\mathcal{S}}, \text{P/E})$, where $\text{VL}^{\mathcal{S}}$ is the voltage array produced by the code-constrained program array $\text{PL}^{\mathcal{S}}$. In this section, we describe our transfer learning approach to achieving this goal.

6.3.1 Review of Generative Flash Modeling

The conditional VAE-GAN architecture underlying the GFM scheme consists of three modules: encoder (Enc), generator (Gen), and discriminator (Dis).

During the training process, the encoder Enc produces latent vectors z from VL based on the VAE technique [62]. The generator Gen reconstructs an array of read voltages, $\widetilde{\text{VL}}$, based on PL, P/E, and z. The P/E vectors are concatenated with the output features of Gen for spatio-temporal combination. The discriminator Dis is trained to distinguish real VL from fake VL.

After optimization, the learned Gen serves as a realistic flash channel simulator which accepts program level array PL, P/E cycle count, and latent vector z as inputs. The latent vector is sampled from a standard multivariate Gaussian distribution. We express the reconstruction of VL in the training and evaluation processes, respectively, as

$$\begin{aligned} &\text{(Train) } \widetilde{\text{VL}} = Gen(\text{PL}, \text{P/E}, Enc(\text{VL})) \\ &\text{(Evaluation) } \widetilde{\text{VL}} = Gen(\text{PL}, \text{P/E}, z). \end{aligned} \tag{6.2}$$

Full details about the training, evaluation, and experimental results are given in [127].

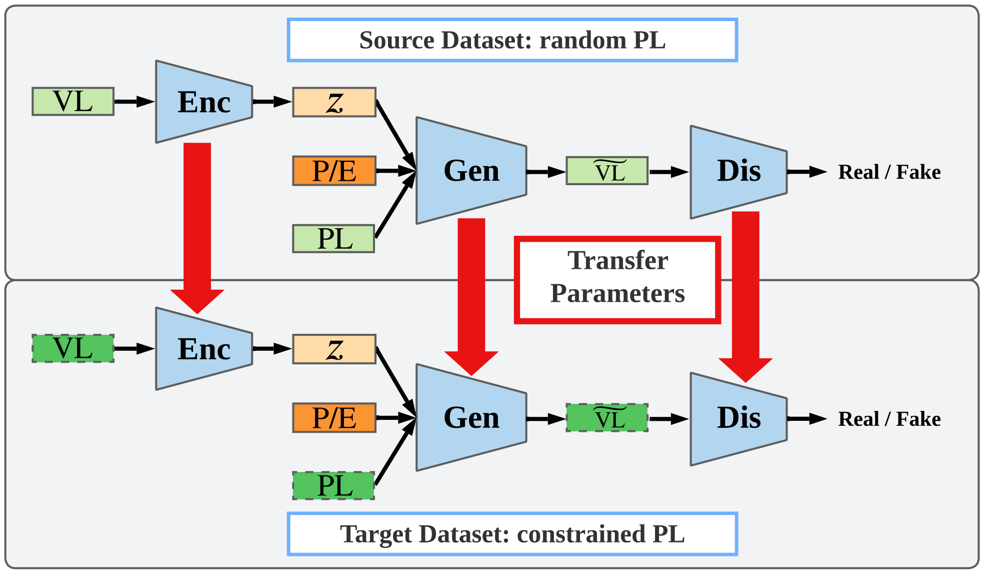
Source Dataset: random PL
VL
Enc
z
P/E
PL
Gen
$\widetilde{\mathrm{VL}}$
Dis
Real / Fake
Transfer Parameters
VL
Enc
z
P/E
PL
Gen
$\widetilde{\mathrm{VL}}$
Dis
Real / Fake
Target Dataset: constrained PL

Figure 6.2. Pipeline of code-aware generative flash modeling.

The GFM approach was demonstrated to accurately reconstruct cell voltage levels by capturing spatial ICI effects and temporal distortions from P/E cycling, as validated by comparing predicted time-dependent and pattern-dependent errors to error measurements.

6.3.2 Code-aware Generative Flash Modeling

The GFM framework is capable of learning the likelihood $P(\mathrm{VL}^{\mathcal{S}}|\mathrm{PL}^{\mathcal{S}}, \mathrm{P/E})$ from a sufficiently large dataset $\{(\mathrm{PL}^{\mathcal{S}}, \mathrm{VL}^{\mathcal{S}}, \mathrm{P/E})\}$ of code-constrained programming measurements at each P/E cycle. To avoid the expense of producing such a large dataset, we propose to use a transfer learning approach. We pre-train the GFM network on a large-scale source dataset $\{(\mathrm{PL}, \mathrm{VL}, \mathrm{P/E})\}$ of VL measurements from pseudo-random (unconstrained) program arrays PL, then fine-tune it using a much smaller target dataset $\{(\mathrm{PL}^{\mathcal{S}}, \mathrm{VL}^{\mathcal{S}}, \mathrm{P/E})\}$ of code-constrained measurements.

We now formulate the pipeline of code-aware GFM. As shown in Fig. 6.2, at the beginning of training, three network modules in GFM (Enc, Gen, and Dis) are initialized with pre-trained weights learned from source dataset. Using the target dataset, the code-aware

GFM follows the framework of GFM to finish the training process. After training, the network parameters in Gen represent the simulator to produce voltage levels from code-constrained PL arrays.

We note that the relation between source and target datasets can impact the transfer learning results. In our case, because random programming arrays very likely include constrained sub-arrays, sharing the pre-trained network weights during the fine-tuning step enables the transfer of relevant knowledge.

6.3.3 Transfer Learning Configuration

It has been observed that pre-training for all network modules can provide better results than pre-training for one individual module [120]. Therefore, in our transfer learning configuration, we share the parameters of all three modules of the pre-trained network.

In our experiments, we consider two read-and-run (RR) constrained codes [39]. The corresponding target datasets $\{(\mathrm{PL}^{\mathcal{S}}, \mathrm{VL}^{\mathcal{S}}, \mathrm{P/E})\}$ consist of pairs of 64×64 PL and VL arrays, collected from a commercial TLC flash device at selected P/E cycles, as in the original GFM setup in [127].

This framework is also applicable to data shaping codes for flash memory [70–72]. These codes minimize the average cell wear due to programming by optimally "shaping" the probability distribution of the programmed cell levels.

The two constrained datasets are collected from a single commercial 1X-nm TLC chip belonging to the same family of chips used for the GFM experiments in [127]. Due to the variation of mappings between manufacturers and product generations, we describe the disallowed patterns of the code-constrained data in terms of the mapping in Fig. 6.1. The first target dataset uses a code constraint $\mathcal{S}_{WL}$ that forbids $\{000, 010\}$ in the lower page of each WL. This eliminates error-prone patterns containing 707, 706, and 607 in the WL direction, as well as other high-low-high error-prone patterns.

The second target dataset uses a code constraint $\mathcal{S}_{2D}$ that forbids $\{000, 010\}$ in lower bits

Table 6.2. Sizes of Training and Evaluation Datasets

	Training	Evaluation
$\lvert\{(\mathrm{PL}, \mathrm{VL}, \mathrm{P/E})\}\rvert$	1.5×10^5	2.1×10^4
$\lvert\{(\mathrm{PL}^{\mathcal{S}_{WL}}, \mathrm{VL}^{\mathcal{S}_{WL}}, \mathrm{P/E})\}\rvert$	1.5×10^4	1.5×10^4
$\lvert\{(\mathrm{PL}^{\mathcal{S}_{2D}}, \mathrm{VL}^{\mathcal{S}_{2D}}, \mathrm{P/E})\}\rvert$	1.5×10^4	1.5×10^4

along both WL and BL directions. This eliminates error-prone patterns containing 707, 706, and 607 in both WLs and BLs, including all patterns shown in Table 6.1, as well as other patterns.

We implement the WL-based constraint $\mathcal{S}_{WL}$ with an interleaved, rate 12:18 run-length limited (RLL) $(d, k) = (0, 1)$ code of overall block length 36 on the lower page, yielding an effective rate of 0.89 [39, 107]. The 2D-constraint is implemented with the 2D RR scheme in [39, 107], which has an effective rate of 0.83.

We collect equal numbers of measured voltages at three P/E cycle counts: 4000, 7000, and 10000. The training and evaluation dataset sizes used in our modeling experiments, described in the next section, are shown in Table 6.2. Note that the size of the target datasets is only 10% of the size of the source dataset.

Remark 5. *In all transfer learning experiments, we use the same settings as were used to train the GFM, namely, batch size 2 and learning rate* 2×10^{-4}. *We settled upon these training parameters after several experiments.*

6.4 Experimental Results and Analysis

In this section, we evaluate our code-aware GFM framework and present results of its application to the two RR constrained codes described in the previous section. We use two evaluation criteria, one to measure the accuracy of the reconstructed results, and the other to measure the training efficiency of the transfer learning procedure. The former is based on probability density functions (PDFs) of the reconstructed voltages, and the latter is based on the number of training iterations required to achieve accurate reconstruction. The evaluation metrics are defined in more detail below.

1. Probability density functions (PDFs): The read voltage PDFs are useful in optimizing read thresholds, gauging cell wear, and estimating bit error rates (BERs). For each P/E cycle, we estimate the conditional PDFs by the frequency of occurrence of measured voltage levels for each given program level. In addition to visually comparing the measured PDFs and reconstructed PDFs, we compute the total variation distance between the two PDFs and compare the associated bit error rates (BERs) on the lower, middle, and upper pages.

2. Training iterations: The number of training iterations needed to achieve satisfactory results can be used as a metric to evaluate the "speed" of the transfer learning process. A training iteration is defined as a single update of the model weights during training. For example, in [127], training the GFM network takes 7 epochs with a batch size of 2 using random programming arrays. With the training dataset size in Table 6.2, the total number of training iterations is 5.25×10^5.

6.4.1 Experimental Settings

We conducted a matrix of experiments to evaluate the effectiveness of transfer learning in code-aware GFM, as summarized in Table 6.3. (See discussion below for an explanation of the abbreviations in the table.) The training iterations are shown in the last column of the table. For convenience, we use a shorthand notation to distinguish the experiments according to the training dataset ("T"), the network initialization ("I"), and the evaluation dataset ("E").

The training dataset corresponded to program arrays based on either pseudo-random data ("T-PR"), $\mathcal{S}_{WL}$-constrained data ("T-WL"), or $\mathcal{S}_{2D}$-constrained data ("T-2D"). Regarding the training mode, training started either from randomly initialized network weights ("I-Rnd") or pre-trained weights ("I-Pre") from T-PR training.

The evaluation mode examined reconstructed voltages generated by pseudo-random data ("E-PR"), $\mathcal{S}_{WL}$-constrained data ("E-WL"), or $\mathcal{S}_{2D}$-constrained data ("E-2D"). Comparisons are made to measurements ("M") from the TLC chip, derived from the pseudo-random dataset ("M-PR"), the $\mathcal{S}_{WL}$-constrained dataset ("M-WL"), or the $\mathcal{S}_{2D}$-constrained dataset ("M-2D").

Table 6.3. Modeling Experiments and Training Iterations

Initialization (I)	Training (T)	Evaluation (E)	Training Iterations
Random	PR	PR,WL,2D	5.25×10^5
	WL	WL	6×10^4
	2D	2D	6.75×10^4
Pre-trained	WL	WL	$\mathbf{7.5 \times 10^3}$
	2D	2D	$\mathbf{7.5 \times 10^3}$

We present results for the following experiments,

1. **M-PR, M-WL, M-2D**: These represent baseline experimental measurements from several 1X-nm flash blocks programmed with pseudo-random, WL-constrained, or 2D-constrained data.

2. **I-Rnd / T-PR / E-(PR,WL,2D)**: We train GFM with random initial network weights using the pseudo-random training dataset and evaluate with pseudo-random data, WL-constrained data, and 2D-constrained data.

3. **I-Pre / T-WL / E-WL**: We initialize GFM with pre-trained weights from the previous training experiment (I-Rnd/T-PR), fine-tune the network using WL-constrained data, and evaluate the model with WL-constrained data.

4. **I-Rnd / T-WL / E-WL**: We train GFM with random initial network weights using the WL-constrained training dataset and evaluate with WL-constrained data.

5. **I-Pre / T-2D / E-2D**: We initialize GFM with pre-trained weights from the first training experiment (I-Rnd/T-PR), fine-tune the network using the 2D-constrained dataset, and evaluate the model with 2D-constrained data.

6. **I-Rnd / T-2D / E-2D**: We train GFM with random initial network weights using the 2D-constrained training dataset and evaluate with 2D-constrained data.

Table 6.4. Total Variation Distance

P/E Cycle Count	4000	7000	10000
$d_{TV}(P_{\text{M-PR}}, P_{\text{I-Rnd/T-PR/E-PR}})$	0.0688	0.0650	0.0687
$d_{TV}(P_{\text{M-WL}}, P_{\text{I-Pre/T-WL/E-WL}})$	0.0696	0.0535	0.0505
$d_{TV}(P_{\text{M-WL}}, P_{\text{I-Rnd/T-WL/E-WL}})$	0.1421	0.1300	0.1020
$d_{TV}(P_{\text{M-WL}}, P_{\text{I-Rnd/T-PR/E-WL}})$	0.1068	0.1116	0.1181
$d_{TV}(P_{\text{M-2D}}, P_{\text{I-Pre/T-2D/E-2D}})$	0.1007	0.0771	0.0908
$d_{TV}(P_{\text{M-2D}}, P_{\text{I-Rnd/T-2D/E-2D}})$	0.1175	0.1021	0.1408
$d_{TV}(P_{\text{M-2D}}, P_{\text{I-Rnd/T-PR/E-2D}})$	0.1470	0.1330	0.1364

6.4.2 PDF Analysis

We now qualitatively and quantitatively analyze the reconstructed voltages from code-aware GFM. First, we visualize the PDFs of the measured and reconstructed read voltages. Fig. 6.3 shows the normalized conditional PDFs of the eight TLC program levels in the reconstructed data for experiment I-Pre/T-WL/E-WL at 7000 P/E cycles. (The plots of voltage PDFs for this experiment at 4000 and 10000 P/E cycles yield qualitatively similar results.) In this log-linear plot, the y-axis represents the probability density and the x-axis represents the read voltages using an arbitrary scale.

Note that the $\mathcal{S}_{WL}$ code constraint on lower pages induces a smaller probability of occurrence for PLs $5, 6, 7$, which is approximately $\frac{1}{3}$ of that of PLs $1, 2, 3, 4$. Qualitatively, the PDFs generated by code-aware GFM (solid curves) closely match the measured PDFs (triangle markers). Similarly, in experiment I-Pre/T-2D/E-2D, the visualization of the model-generated PDFs accurately reflects the measured PDFs and their dependence on P/E cycles.

Next, we evaluate the PDF results of the code-aware GFM experiments quantitatively using total variation (TV) distance, d_{TV}. This distance provides a measure of the difference between the real (measured) distributions P_{real} and the fake (reconstructed) distributions P_{fake},

$$d_{\text{TV}}(P_{real}, P_{fake}) = \frac{1}{2}\sum_{\text{VL}} |P_{real}(\text{VL}) - P_{fake}(\text{VL})|. \tag{6.3}$$

The numerical results are shown in Table 6.4. We find that pre-training helps code-aware GFM

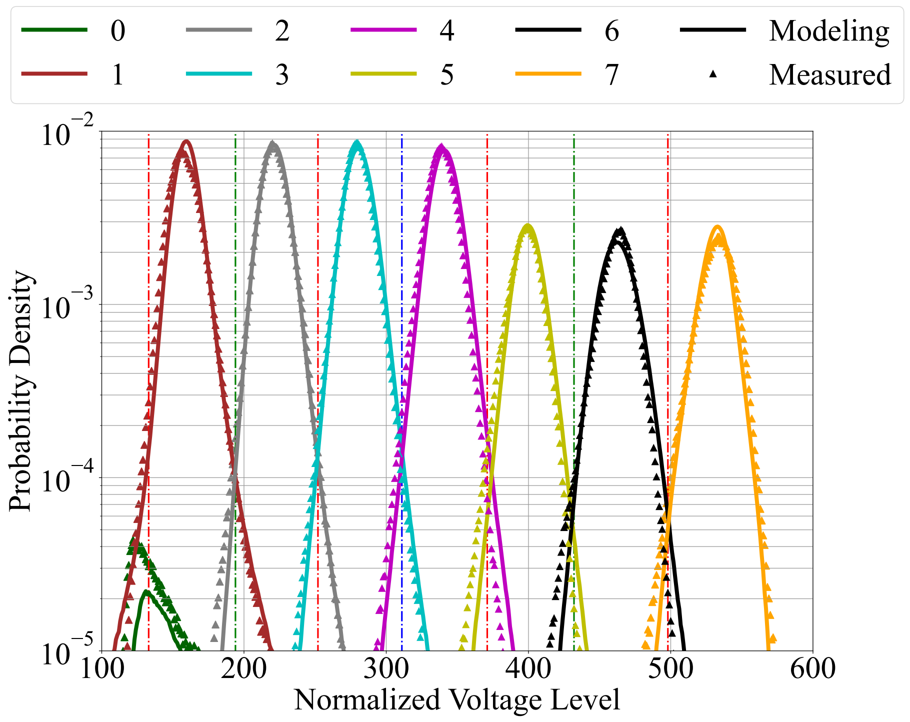

Figure 6.3. PDF plots in logarithmic scale for measured and regenerated voltage levels (experiment I-Pre/T-WL/E-WL) at 7000 P/E cycles. The visualization is based on dataset $\{(\mathrm{PL}^{\mathcal{S}_{WL}}, \mathrm{VL}^{\mathcal{S}_{WL}}, \mathrm{P/E})\}$.

produce distributions with the least TV distance in both $\mathcal{S}_{WL}$-coded and $\mathcal{S}_{2D}$-coded scenarios.

It is also important to consider the tails of the distributions, which have a major impact on the channel error rate. As discussed in Section 6.2.1, cell level errors are determined by comparing the read voltages to the read thresholds, and the resulting bit errors on pages arise from the mapping between cell levels and their corresponding 3-bit binary logic values. We compared measured and reconstructed page bit error rates (BERs) from the ten experiments described in Section 6.4.1. The results are shown in Fig. 6.4.

The leftmost three sub-figures in Fig. 6.4 pertain to the lower, middle, and upper pages in the $\mathcal{S}_{WL}$-coded case, respectively. The six curves in each plot correspond to the experimental measurements M-PR and M-WL, the GFM modeling experiments I-Rnd/T-PR/E-PR and I-Rnd/T-PR/E-WL, and the code-aware GFM experiments I-Rnd/T-WL/E-WL and I-Pre/T-WL/E-WL using the $\mathcal{S}_{WL}$ dataset, comparing training "from scratch" and with pre-trained network

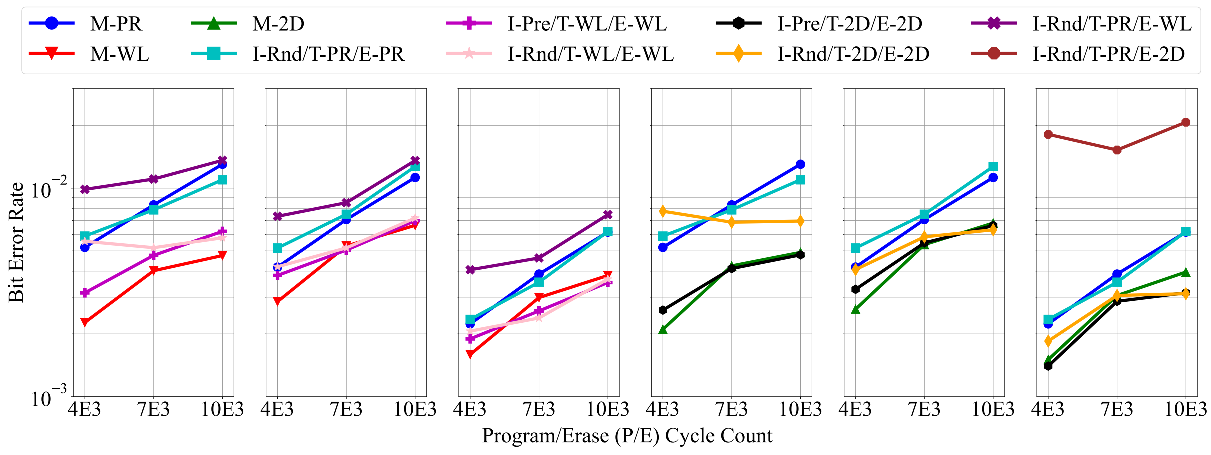

Figure 6.4. BER comparisons: the leftmost (resp., rightmost) three sub-figures show lower, middle, and upper page BERs for $\mathcal{S}_{WL}$-coded (resp., $\mathcal{S}_{2D}$-coded) data.

parameters.

The M-PR and M-WL curves show that $\mathcal{S}_{WL}$ coding decreases the measured BER on all three pages at all three measured P/E cycles, confirming the observations in [39]. The GFM experiment I-Rnd/T-PR/E-PR using random initialization along with training and evaluation on pseudo-random data reconstructs page BERs quite accurately at all three P/E cycles, a finding that is consistent with [127]. However, when this GFM network is evaluated using the $\mathcal{S}_{WL}$-coded dataset in experiment I-Rnd/T-PR/E-WL, we see that the reconstructed BERs are significantly higher than the measured BERs in the M-WL curve at all three P/E cycles. This suggests that the pseudo-random dataset does not sufficiently capture all of the characteristics of the coded channel.

The final two curves compare the effects of random initialization and pre-training in the code-aware GFM networks obtained by training and evaluating on the $\mathcal{S}_{WL}$ dataset. We see that the two experiments yield very similar reconstructed BERs, with the exception of the lower page BER at 4000 P/E cycles, where random initialization yields a noticeably more inaccurate estimate. Overall, the reconstructed BERs qualitatively track the measured M-WL results reasonably well, although both models overestimate BER in lower pages at all P/E cycles, as well as in middle pages at 4000 P/E cycles.

The rightmost three sub-figures in Fig. 6.4 show the corresponding BER results for lower, middle, and upper pages in the $\mathcal{S}_{SD}$-coded case, respectively. (The BERs of I-Rnd/T-PR/E-2D for lower and middle pages are at least 3×10^{-2}; thus, the curves are not shown in the sub-figures.) The overall conclusions drawn from these curves are similar to the $\mathcal{S}_{WL}$-coded case, although we see that the GFM trained on the pseudo-random source dataset does an even worse job of learning the $\mathcal{S}_{2D}$-coded channel.

6.4.3 Iteration Number Analysis

The number of training iterations used in the experiments was determined by comparing the reconstructed PDFs to the corresponding measured PDFs using TV distance.

From Table 6.3, we find that the number of iterations required to fine-tune the code-aware GFM network from the pre-trained model, 7.5×10^3, is only 12.5% (resp., 11.11%) of the number required when training from scratch using the target $\mathcal{S}_{WL}$ (resp., $\mathcal{S}_{SD}$) dataset, namely 6×10^4 (resp., 6.75×10^4).

Specifically, when training from scratch using the smaller target dataset, we observed that in the early training iterations the reconstructed read voltage PDFs do not accurately capture temporal P/E cycle variations and tail behavior. On the other hand, adaptation from a single GFM network pre-trained with a sufficiently large source dataset of pseudo-random data provides enough channel knowledge to significantly accelerate the learning process from both of the smaller target datasets.

6.5 Conclusion

This chapter presents an application of transfer learning to generative modeling of read voltages in flash memory channels. We fine-tune a generative model pre-trained with a large source dataset of pseudo-random spatio-temporal data using much smaller code-constrained target datasets. By comparing measured and reconstructed read voltage probability distribution functions and page bit error rates in a commercial TLC flash memory, we demonstrate that pre-training can accelerate learning for multiple generative modeling tasks even when the amount of target training data is very limited. These results motivate further investigation into the use of transfer learning in applications of machine learning to data storage and communication systems.

Acknowledgement

This chapter is in part a reprint of the material in the paper [130]: Simeng Zheng and Paul H. Siegel, “Code-aware storage channel modeling via machine learning,” in *Proc. IEEE Information Theory Workshop (ITW)*, Mumbai, India, Nov. 2022, pp. 196–201. The work was presented at ITW 2022 [130]. The dissertation author would like to thank the collaborator in this publication.

Chapter 7

PR-NN: RNN-based Detection for Coded Partial-Response Channels

7.1 Introduction

7.1.1 Background on magnetic recording

The read/write process in longitudinal magnetic recording is often modeled as a continuous-time, linear time-invariant (LTI) system with bipolar input waveforms taking values -1 and $+1$ on time intervals of fixed length T_c. The step response often takes the form of a unimodal pulse with a finite pulsewidth at half the maximum amplitude (PW50) whose size relative to the channel input interval ($PW50/T_c$) roughly determines the extent of inter-symbol interference (ISI) arising from adjacent transitions in the input waveform. When the channel output signals are synchronously sampled at intervals of T_c, the sampled system is often modeled as a finite impulse-response discrete-time LTI system [115].

In order to facilitate the recovery of the input signal, magnetic recording systems often use some form of partial-response (PR) equalization. The PR equalizer shapes the readback signal in such a way that only a finite number of values are observed at sample times. For a range of linear recording densities, the sampled Lorentzian output response of the PR-equalized longitudinal recording channel is well modeled by the family of extended PR “class 4” (PR4) channels, denoted by $\mathrm{E}^{\mathrm{N-1}}$PR4 [113]. The impulse response of the $\mathrm{E}^{\mathrm{N-1}}$PR4 channel can be represented by $x_N(D) = (1 - D)(1 + D)^N$, where D is the delay operator and N is a positive integer. The channel can be treated as a linear filter with integer coefficients, whose outputs are generated by a linear finite-state machine, where the number of states is 2^{N+1}. In practice, the selected PR step response is governed by the channel step response and the choice of $PW50/T_c$. When sampled, the PR-equalized noisy magnetic recording channel is often modeled, to first order, as a linear finite-state machine with additive, correlated noise.

The sampled PR-equalized magnetic recording channel resembles a digital communication channel, and suitable detection and coding methods from communication theory can be beneficially applied. The finite-state structure of the ISI channel is amenable to trellis-based sequence detection methods such as Viterbi detection [117], which is optimal if the additive noise is assumed to be white Gaussian noise (AWGN). The combination of PR channel equalization and Viterbi detection is referred to as PRML, an acronym for "partial-response (PR) equalization with maximum-likelihood (ML) sequence detection" [20, 54]. Channels can also use maximum a-posteriori probability (MAP) symbol detection based upon, for example, the BCJR algorithm [4], which is also optimal in AWGN. When combined with PR equalization, the resulting system is referred to as PRMAP. To account for noise correlation (colored noise) due to equalization, Noise-Predictive Maximum Likelihood (NPML) detectors embed a noise prediction/whitening process into the branch metric computation of a Viterbi detector [21]. In longitudinal recording systems, NPML detectors offer significant performance gains over PRML detectors [21, 85].

Magnetic recording systems often use both an error-correcting code and a constrained code. The general purpose of a constrained code is to improve the performance of the system by matching the characteristics of the recorded signals to those of the channel [49]. In the context of PRML-type systems, the constrained code is often used to improve timing recovery as well as to increase the distinguishability of the sampled output sequences. Several classes of such "distance-enhancing codes" have been proposed, such as runlength-limited (RLL) codes [86] and matched spectral null (MSN) codes [55].

In a coded PRML-type system, equalization plays a crucial role in determining the performance of the system. Advanced detectors must take into account the noise correlation and signal misequalization effects due to equalization [125]. The channel parameters in a magnetic hard disk storage system can vary due to several factors, including variations in temperature, head flying-height, and track-dependent rotational speed [122]. These parameter variations need to be taken into consideration in the design of the PR equalizer and they can also influence the performance of the detector. In this project, we explore the use of machine learning methods – specifically recurrent neural networks (RNNs) – to design robust detectors for magnetic recording systems.

7.1.2 Machine learning for coded communication

In recent years, machine learning has demonstrated its effectiveness in a wide range of applications, specifically in the fields of computer vision and natural language processing. The huge success of machine learning in these areas has triggered the interest of researchers to apply deep learning (DL) methods and neural networks (NNs) to channel coding problems. Nachmani *et al.* proposed *Weighted Belief Propagation* (WBP) algorithm using deep neural networks (DNNs) to decode noisy linear codewords [89]. This approach was further studied by Lian *et al.* in the context of simple scaling models with reduced complexity [68]. In [101], Satorras and Welling considered a hybrid model that combines belief propagation with an extension of graph nerual networks to factor graphs (FG-GNNs). Kim *et al.* presented the first end-to-end communication system for feedback channels designed using deep learning, with RNN models for encoding and decoding [59]. Jiang *et al.* incorporated some aspects of an iterative turbo decoder into an RNN-based end-to-end machine learning architecture that provides robust, near-optimal recovery of noisy turbo codewords, without BCJR knowledge [52]. Shlezinger *et al.* introduced ViterbiNet, a decoder that incorporates DNNs into the channel state information (CSI) estimate of the Viterbi algorithm [106]. They also invented BCJRNet, a detection approach that implements data-driven BCJR MAP symbol detector [105].

With all of these learned communication systems, the length of codewords is quite limited because the training complexity grows exponentially in the length [32]. Farsad and Goldsmith addressed this problem by creating a sliding bidirectional RNN to process a longer signal stream [26]. Bennatan *et al.* decoded codewords of an arbitrary block length by extracting the syndrome of the hard decisions and the channel output reliabilities [7]. Tandler *et al.* described a training method that gradually introduces code sequences with an increasing number of ones to limit the complexity, and used it to recover long convolutional codewords [110]. Nevertheless, during the evaluation stage, these NN-based decoders are not well suited to handling continuous streaming data, which would typically be produced by convolutional encoders and ISI channels.

7.1.3 Our Contribution

In this work, we propose a novel NN architecture for detection of input-constrained PR-equalized magnetic recording channels, which we refer to as *partial-response neural network* (PR-NN). The PR-NN detector is designed for application to continuous, streaming channel outputs. The sequential processing properties of RNN cells [17, 19], including gated recurrent units (GRUs) [18] and long short-term memory (LSTM) [44], are naturally suited to the time-dependent outputs from PR channels. The primary component of our PR-NN architecture is a bi-directional gated recurrent unit (bi-GRU). (We settled on a GRU-based architecture after initial experiments indicated superior performance compared to an LSTM network. These results are not included here.)

We train and evaluate the PR-NN under various scenarios: ideal PR channel outputs with AWGN, ideal PR channel outputs with additive colored noise (ACN) generated by a minimum mean squared error (MMSE) equalizer for the Lorentzian channel, and MMSE-equalized Lorentzian outputs with corresponding equalized noise. By training the model under multiple scenarios, we show that a single PR-NN detector can be used as a substitute for multiple classical detectors. Moreover, training over a range of channel signal-to-noise ratios (SNRs) and channel densities allows the PR-NN to adapt to a variety of channel conditions. Special training and evaluation techniques make the PR-NN compatible with detection of continuous, streaming data, with no constraint on sequence length, thus overcoming a key limitation of previous NN-based decoding strategies. The continuous decoding relies on a *sliding-window* evaluation process. We also show that the computational complexity of the PR-NN detector is comparable with that of Viterbi detection.

We conduct our experiments using the E^{2}PR4 channel model, which matches the characteristics of the Lorentzian channel for high recording densities. The binary system inputs are constrained by a rate-2/3, (1,7)-RLL constrained sliding-block decodable finite-state encoder [119]. The constrained codewords are mapped into binary channel inputs by a non-return-to-zero-

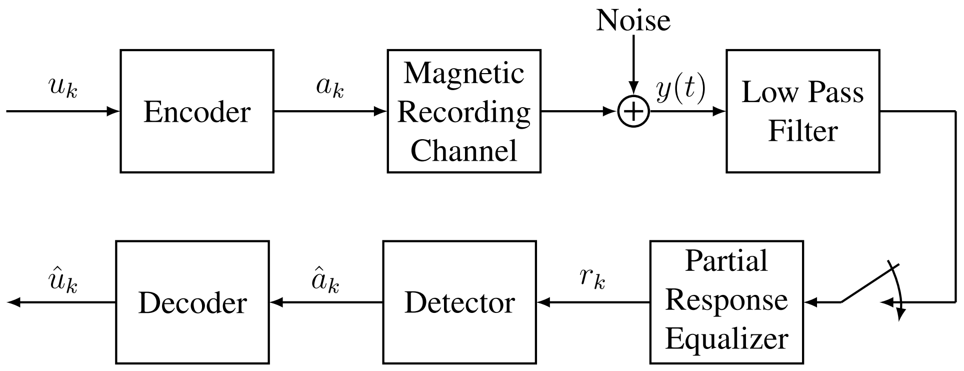

Figure 7.1. System architecture.

inverse (NRZI) precoder with system function represented by $c(D) = 1/(1 + D)$. The resulting sequence is modulated to bipolar form to represent the magnetization pattern corresponding to the recording channel input [49]. With AWGN and a reduced-state Viterbi detector that reflects the input constraint, the system serving as our benchmark achieves a 2.2dB coding gain over the uncoded E^{2}PR4 system [6].

The bit error rate (BER) performance of the PR-NN detector compares favorably to that of the classical detectors – PRML, PRMAP and NPML – in the scenarios where they are known to perform well. More importantly, the PR-NN detector exhibits a robustness not shared by the other detectors when it is jointly trained in multiple scenarios. In fact, under joint training, PR-NN essentially maintains the performance that is achieved with separate training. These results suggest that robust detection architectures like PR-NN may hold promise for application in practical recording systems.

7.2 System Architecture and Detectors

7.2.1 System model

A block diagram of a magnetic-disk recording system is shown in Fig. 7.1. User data $\{u_k\}$ ($u_k \in \{0, 1\}$) is encoded using a (d, k) run-length limited (RLL) code [49]. The constrained codewords are then precoded and mapped into the symbol sequence $\{a_k\}$ ($a_k \in \{-1, +1\}$).

The precoder maps a binary sequence to the two-level channel input sequence. The precoding convention we use is nonreturn-to-zero-inverse (NRZI), where the precoder has system function $c(D) = 1/(1 + D)$. The modulation method is binary phase shift keying (BPSK), where 0 maps to -1 and 1 maps to $+1$.

During the write process, the two-level channel input is converted into a one-dimensional magnetization pattern on the magnetic medium in the disk. The disk spins with controlled speed, and the read and write heads effectively rotate over a track on the surface of the magnetic medium [122]. During the read process, the disk rotates beneath the read head, which senses the magnetic field induced by the magnetization pattern along the track. The read-back signal can be regarded as a linear superposition of the dipulse response corresponding to a positive pulse of width equal to a single channel bit duration at the input to the channel. Mathematically, the read-back signal $y(t)$ can be expressed as

$$y(t) = \sum_{i=-\infty}^{\infty} a_i q(t - iT_c) + \eta(t) \tag{7.1}$$

where T_c is the channel bit spacing, and sequence $\{a_k\}$ is written on the disk at a rate of $1/T_c$. The *dipulse response* $q(t)$ is expressed as $q(t) = g(t) - g(t - T_c)$, where $g(t)$ is the unit step response of the channel. The term $\eta(t)$ represents the additive white Gaussian noise process.

The function $g(t)$ is often called the transition response of the recording system, and its characteristics are related to the specific design of the recording heads and magnetic medium. Recording systems are typically classified into two types: longitudinal [84] and perpendicular [122]. The Lorentzian model for the transition response is commonly used for longitudinal recording systems. The $tanh$ function and error function approximation are widely used for perpendicular recording systems. In this work, we focus on longitudinal recording with Lorentzian transition response

$$g(t) = \frac{1}{1 + (2t/PW_{50})^2} \tag{7.2}$$

where PW_{50} is the single parameter of the Lorentzian model and denotes the pulsewidth at 50% maximum amplitude. The recording density is characterized by the normalized density parameter PW_{50}/T_c.

The noisy channel output signal passes through a low pass filter (LPF). The filtered signal is sampled at the rate $1/T_c$, generating samples at times $t = kT_c$. The samples are filtered by a discrete-time equalizer which is designed to optimize detector performance. The most common scheme used for equalization and detection in longitudinal recording systems is partial-response maximum-likelihood detection (PRML). In this scheme, a *finite impulse response* (FIR) equalizer is designed to equalize the channel response to a relatively short-duration partial-response (PR) target, and the channel input sequence is recovered from the equalized signal by a maximum-likelihood detector based on the Viterbi algorithm. The family of equalizers called "Class-4" and "extended Class-4" are often used in longitudinal magnetic recording, where the choice of equalizer target is matched to the channel density [113]. The general expression for the samples of the target equalized dipulse response, expressed as a D-transform polynomial, takes the form

$$\begin{aligned} x(D) &= (1-D)\alpha(D) = (1-D)(1+D)^N \\ &= x_0 + x_1 D + \cdots + x_{N+1} D^{N+1} \end{aligned} \tag{7.3}$$

where $\alpha(D) = \alpha_0 + \alpha_1 D + \cdots + \alpha_N D^N$. When $N = 1$, the channel is called a Partial-Response Class-4 (PR4) channel. When $N \geq 2$, the channels are call Extended Partial-Response Class-4, denoted individually as $\mathrm{E}^{\mathrm{N-1}}$PR4.

There are many papers that address the design of the PR equalizer, such as [54, 87]. A common design objective is the MMSE equalizer, which minimizes the mean squared error of the target PR signal and the equalized channel output. Since the channel parameters may vary across and even along tracks on a magnetic disk, the equalizer can be designed to be adaptive to the channel properties. Some adaptive equalization architectures for PR channels can be found in [115].

In the longitudinal recording system, if the target PR signal is chosen to be E$^{\text{N-1}}$PR4 signal, the coefficients of the PR equalizer can be optimized to achieve an overall transfer function that reflects the head/medium characteristics and the analog LPF frequency response. If we assume an ideal LPF, the equalizer coefficients $\{z_i\}$ can be specified as

$$\begin{aligned} z_i &= \frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{x(e^{-j\omega})}{Q(\omega)} e^{j\omega i} d\omega \\ &= \frac{1}{\pi^2}\sum_{\ell=0}^{N} \alpha_i \frac{(-1)^{\ell} e^{\pi PW_{50}/2}\cos(i\pi) - PW_{50}/2}{(PW_{50}/2)^2 + (i-\ell)^2} \end{aligned} \tag{7.4}$$

where $Q(\omega)$ is the frequency response of the Lorentzian channel and $T_c = 1$ [84].

Thus the output r_k of the PR equalizer in Fig. 7.1 consists of an ideal PR signal plus an additive distortion. Mathematically, the equalizer output r_k can be written as

$$r_k = \sum_{i=0}^{N+1} x_i a_{k-i} + n_k \tag{7.5}$$

where $\{x_i\}$ are the coefficients of the target E$^{\text{N-1}}$PR4 channel and $\{n_k\}$ denotes the additive distortion. The distortion n_k can be decomposed as

$$n_k = \sum_i z_i \eta_{k-i} + (\sum_i \tilde{q}_i a_{k-i} - \sum_{i=0}^{N+1} x_i a_{k-i}) \tag{7.6}$$

where the summation represents additive colored noise corresponding to the equalized samples of the low-pass filtered white noise and the expression in parentheses represents the misequalization error. Here $\{\tilde{q}_i\}$ correspond to the convolution of the PR equalizer taps with the sampled channel dipulse response.

As discussed above, given a PR target, a trellis-based Viterbi detector [117] is used to detect the data sequence from the noisy channel output sequence $\mathbf{r}$. For the Viterbi detector, the branch metric calculation is based on the squared-Euclidean distance between the noisy channel output sample and the targeted PR channel output sample labeling the particular branch. The

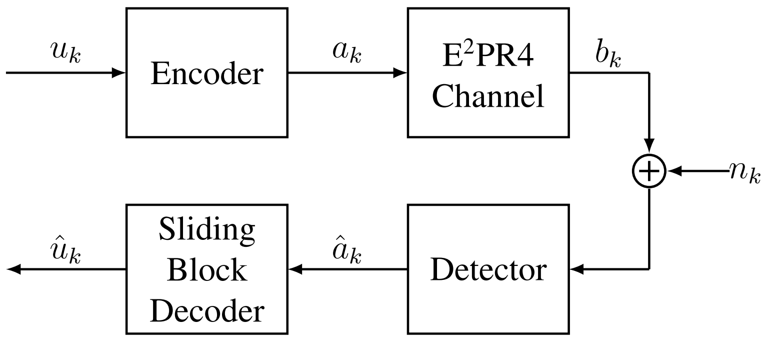

Figure 7.2. Discrete model for simulation, where n_k represents the additive distortion.

combination of PR equalization with Viterbi detection is called PRML. The BCJR detector [4], which is based upon a MAP symbol detection algorithm, is also of interest for data recovery. For the BCJR detector, the *log-likelihood ratios* (LLRs) can be derived from forward recursion, backward recursion, and branch transition probability computations. The system combining PR equalization with BCJR detection is called PRMAP.

As shown in (7.6), the noise in the longitudinal recording system model is composed of colored noise and misequalization error, which cannot be modeled as AWGN. Therefore, the Viterbi detector is not an optimal sequence detector. NPML detection [21] combines a linear noise prediction/whitening filter with Viterbi detection. The coefficients of the noise predictor are designed to minimize the mean squared error (MSE) of the noise and the predictor output. Algorithmic details of these detection methods will be formulated in Section 7.2.3.

The decoder that recovers user data estimates $\hat{u}_k$ from channel input estimates $\hat{a}_k$ is implemented by means of a sliding-block decoder (which is here assumed to incorporate a $1 + D$ post-coder operation). The decoder has memory m and anticipation a, meaning that the current detected codeword, the previous m detected codewords, and the following a detected codewords are all used to determine the corresponding current user data word [80].

7.2.2 Digital channel implementation

In the following, we formulate the digital implementation of the magnetic recording system, as shown in Fig. 7.1, that we will use for our simulations. The outputs of the PR equalizer can be modeled as the outputs of a binary-input, linear ISI channel with additive distortion as derived in (7.5) and (7.6). In particular, we can model the ISI channel resulting from the magnetic recording channel, the LPF, the sampler and the PR equalizer in Fig. 7.1 as an $\mathrm{E}^{\mathrm{N-1}}$PR4 channel, as shown in Fig. 7.2. For our experiments, we focus on the specific case of $N = 3$, namely E^2PR4, as this was widely used in practice. The outputs of the channel model can then be represented as

$$r_k = b_k + n_k = \sum_{i=0}^{4} x_i a_{k-i} + n_k \tag{7.7}$$

where $x_0 = 1$, $x_1 = 2$, $x_2 = 0$, $x_3 = -2$ and $x_4 = -1$. Here b_k denotes the noiseless output from the E^2PR4 channel and a_k denotes the channel input.

We now describe the finite-state channel representation of the input-constrained E^2PR4 channel. In order to improve the performance of PRML-type systems, several classes of codes with distance-enhancing properties have been proposed, such as forbidden list codes [49] and *matched spectral null* (MSN) codes [55]. In [6], Behrens and Armstrong show that $(1, \infty)$-RLL constrained codes provide a coding gain when applied to the E^2PR4 channel. To see this, note that a sequence satisfies the $(1, \infty)$-RLL constraint if the runs of 0s between successive 1s have length at least 1 [80]. In other words, consecutive 1s are forbidden. This means that, in the precoded $(1, \infty)$-RLL sequence, the strings 101 and 010 are prohibited. The minimum squared-Euclidean distance between channel output sequences corresponding to a closed error event (paths in the detector trellis that agree except on a finite number of branches) is 6. These events correspond to the channel inputs $+1\ -1\ +1$ and $-1\ +1\ -1$. On the other hand, for the $(1, \infty)$ input-constrained channel, with these channel inputs forbidden, the minimum distance associated with a closed error event increases to 10. (A more detailed discussion of this sort of distance analysis is found in [55].) This offers the possibility of an effective 2.2 dB gain in signal-to-noise ratio (SNR), ignoring the rate loss associated with the use of the constrained code, provided that the detector trellis is modified to reflect the constraints. Note that this gain also applies to the (1,7)-RLL input-constrained channel.

Incorporating the $(1, \infty)$-RLL constraint into the Viterbi detector for the E^{2}PR4 channel not only eliminates the dominant error events, but also reduces the required number of states in the detector trellis (i.e., the number of states realized by the channel finite state machine) from 16 to 10. To see this, for convenience, we ignore the BPSK modulation used to generate the bipolar channel inputs and, by a slight abuse of notation, we let $a_k \in \{0, 1\}$ denote the channel inputs. At time k, channel state transitions from state $\mathbf{s}_{k-1} = (a_{k-4}a_{k-3}a_{k-2}a_{k-1})$ to state $\mathbf{s}_k = (a_{k-3}a_{k-2}a_{k-1}a_k)$ with associated output b_k. This is represented in the channel state machine diagram by an edge from state $\mathbf{s}_{k-1}$ to state $\mathbf{s}_k$ with input/output label a_k/b_k. When the $(1, \infty)$-RLL constraint is applied, states (0010), (0100), (0101), (1010), (1011) and (1101) are eliminated, along with all their incoming and outgoing branches because they represent violations of the constraint. The resulting state machine diagram for the input-constrained E^{2}PR4 channel is shown in Fig. 7.3. This reduced state machine provides the structure of the trellis that can be used at each time step of the reduced-state Viterbi detector.

We selected the (1,7)-RLL constraint for our system since it has been widely used in commercial magnetic tape and hard disk recording systems. For the encoder and decoder, we use the rate-2/3 Weathers-Wolf code [119], which achieves the minimum possible number of states for any rate-2/3 (1,7)-RLL code. The code is $(0, 2)$-sliding-block decodable, meaning that the decoding algorithm can be implemented by a sliding-block decoder, where the current (length-3) codeword along with the following two codewords are used to determine the corresponding (length-2) input word. The encoder and decoder structures are given in [119]. In the sliding-block decoder, a single channel bit error can affect the decoding of up to 3 input words, or 6 user bits, so the error propagation is limited.

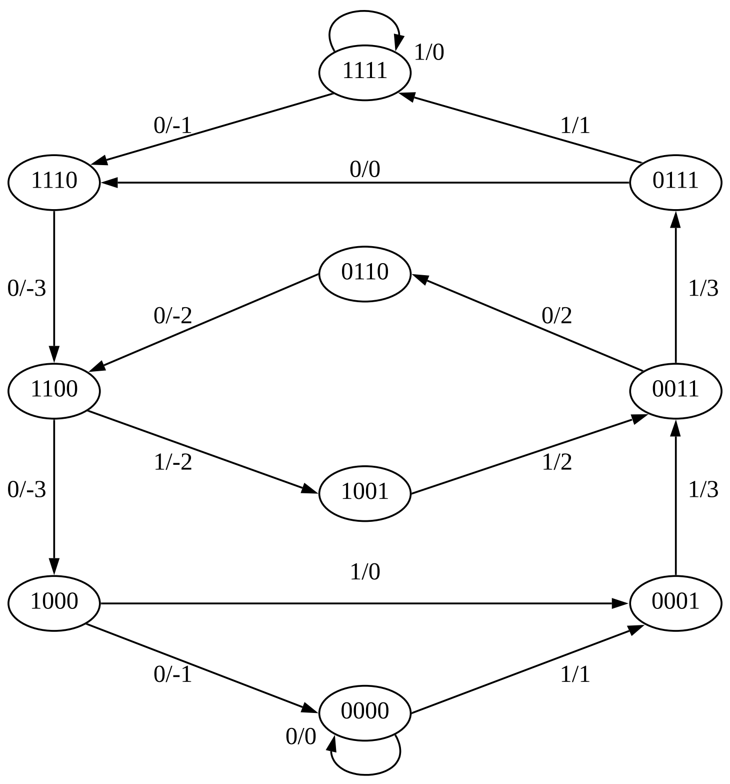

Figure 7.3. $(1, \infty)$-RLL input-constrained E^{2}PR4 channel state machine.

7.2.3 Signal Detection Methods

In the following, we review three classical signal detection methods for the magnetic recording system. Given the noisy sequence $\mathbf{r}$, the detector will output an estimate $\hat{\mathbf{a}}$ of the channel input sequence.

1. Viterbi detection: When we incorporate the $(1, \infty)$-RLL constraint into the E^{2}PR4 state machine, the 10-state graph determines the trellis structure for the Viterbi detector. The Viterbi detector maximizes the likelihood (conditional probability) Pr$(\mathbf{r}|\mathbf{a})$ [117]. When the noise is AWGN, the branch metric is the squared Euclidean distance. Specifically, the branch metric at time kT_c from state $\mathbf{s}_j$ to state $\mathbf{s}_m$ takes the form

$$\lambda_k(\mathbf{s}_j, \mathbf{s}_m) = [r_k - (a_k(\mathbf{s}_m) + \sum_{i=1}^{4} x_i a_{k-i}(\mathbf{s}_j))]^2 \tag{7.8}$$

where $a_k(\mathbf{s}_m)$, $a_{k-1}(\mathbf{s}_j)$, $a_{k-2}(\mathbf{s}_j)$, $a_{k-3}(\mathbf{s}_j)$ and $a_{k-4}(\mathbf{s}_j)$ are the BPSK input values determined by hypothesized state transition $\mathbf{s}_j \rightarrow \mathbf{s}_m$.

2. BCJR detection: The BCJR detection algorithm maximizes the a-posteriori probability $\Pr(\mathbf{a}|\mathbf{r})$. The complete derivation can be found in [4]. In order to reduce the computational complexity, we make use of a modified algorithm, called *max-log-map* detection, that uses the approximation ln $\sum_j e^{a_j} \approx \max_j a_j$.

3. NPML detection: As mentioned above, the additive distortion in a realistic longitudinal recording system cannot be considered to be simply AWGN. A better approximation takes into account the noise coloration introduced by the equalizer as well as the misequalization error. In the presence of such noise, the Viterbi detector using squared Euclidean distance metric will not provide optimal detection and the system performance will be degraded [85]. NPML detection introduces a noise prediction process into the branch computation of the Viterbi detector that significantly improves the system performance [21]. An estimate of the current noise sample, $\hat{n}_k$ is formed from previous N_p noise samples, and then subtracted from r_k. The coefficients of the N_p-tap noise predictor $\{p_i\}$ are chosen to minimize the mean squared error between the noise n_k and the estimate $\hat{n}_k$,

   $$\mathbb{E}[|n_k - \hat{n}_k|^2] = \mathbb{E}[|n_k - \sum_{i=1}^{N_p} n_{k-i} p_i|^2], \tag{7.9}$$

   where n_k takes the form in (7.6). The derivation of the MMSE predictor coefficients can be found in [21].

   The implementation of the NPML detector requires the use of tentative decisions from the survivor path memory associated with each state of the Viterbi detector. Mathematically, the branch metric at time kT_c from state $\mathbf{s}_j$ to state $\mathbf{s}_m$ takes the form

$$\begin{aligned}\lambda_k(\mathbf{s}_j, \mathbf{s}_m) = [r_k - \sum_{i=1}^{N_p}(r_{k-i} - (\sum_{l=0}^{4} x_i \hat{a}_{k-i-l}(\mathbf{s}_j)))p_i \\ - (a_k(\mathbf{s}_m) + \sum_{i=1}^{4} x_i a_{k-i}(\mathbf{s}_j))]^2\end{aligned} \tag{7.10}$$

where the terms $\hat{a}_{k-i}(\mathbf{s}_j)$, $\hat{a}_{k-i-1}(\mathbf{s}_j)$, $\hat{a}_{k-i-2}(\mathbf{s}_j)$, $\hat{a}_{k-i-3}(\mathbf{s}_j)$ and $\hat{a}_{k-i-4}(\mathbf{s}_j)$ represent past decisions taken from the survivor path history associated with state $\mathbf{s}_j$, and $a_k(\mathbf{s}_m)$, $a_{k-1}(\mathbf{s}_j)$, $a_{k-2}(\mathbf{s}_j)$, $a_{k-3}(\mathbf{s}_j)$ and $a_{k-4}(\mathbf{s}_j)$ are determined by the hypothesized state transition $\mathbf{s}_j \to \mathbf{s}_m$.

7.2.4 Detector implementation details

In practice, Viterbi detectors can retain only a finite path memory and must make an output decision after some fixed delay whether or not all survivor paths have merged. A common practice is to determine the trellis state with the minimum survivor path metric, and then to trace back along the path to the initial branch, whose label is then used to generate the estimated input/output symbol (or word, in the case of a convolutional code). Various estimates for a suitable traceback length L_{tb} for convolutional codes have been proposed, based upon random coding analysis [29], code sequence properties [42], and experimentation [82, 83]. A reasonable rule of thumb for a rate-r code with memory ν is

$$L_{tb} \approx A\frac{\nu}{1-r} \tag{7.11}$$

where A is between 2 and 3. For rate $r = 1/2$ codes, this agrees with the often cited estimate $L_{tb} \approx A\nu$ with A between 4 and 6. Similar methods have been used to estimate L_{tb} for Viterbi detection of ISI channels, and a reasonable choice for the traceback length, which we use to guide our experiments, is $L_{tb} \approx 5\nu$.

Rather than decoding one symbol at each iteration of the survivor metric update procedure in the Viterbi detector, we will make use of a sliding-window approach. In the sliding-window decoder, successive blocks of a specified "evaluation length" L_{eval} are estimated, as illustrated schematically in Fig. 7.4. The survivor metric computation starts at time 0, and the survivor update procedure is performed continuously as the first $L_{eval} + L_{overlap}$ received symbols arrive, where $L_{overlap} \geq L_{tb}$. The detector then traces back along the survivor path corresponding to the state with the smallest survivor metric. The symbol estimates along the first L_{eval} branches can be considered to be fairly reliable and the detector outputs these values. The first L_{eval} branches of the survivor paths are then truncated, and the detector proceeds to extend the remaining portion of the survivor paths for another L_{eval} steps, up to time $2L_{eval} + L_{overlap}$. The detector then traces back along the minimum metric survivor path and outputs the symbol estimates along the first L_{eval} branches, corresponding to symbols at time $L_{eval} + 1$ through $2L_{eval}$. This process is then repeated. In each successive block from time mL_{eval} to time $(m+1)L_{eval} + L_{overlap}$, we refer to the first L_{eval} steps as the evaluation part and the final $L_{overlap}$ steps as the overlap part. The final $L_{overlap}$ symbols can be treated as dummy symbols, or a termination sequence of dummy symbols can be used to force the detector to a known state, enhancing the reliability of the last group of estimated symbols. We adopt the latter termination approach in our simulations.

The sliding-window approach is easily adapted to NPML detection. For the BCJR detector, a conceptually similar sliding-window approach can be implemented using a forward state metric processor and a pair of backward state metric processors [116].

The length of the evaluation block, L_{eval}, can be chosen to be any size greater than or equal to one symbol, with the lower limit corresponding to conventional symbol-by-symbol Viterbi decoding. Larger sizes increase the required storage for survivor paths and the delay until the first symbol is decoded. However, the use of longer survivor paths should increase the likelihood of survivor path merging, thereby improving reliability of decoding. In Section 7.3, where we adopt a similar block streaming decoding architecture, the sizes of L_{eval} and $L_{overlap}$ have an exponential effect on the size of the training dataset. This plays a role in the choice of these parameters.

Remark 6. *For the digital implementation of the longitudinal recording channel, we assume*

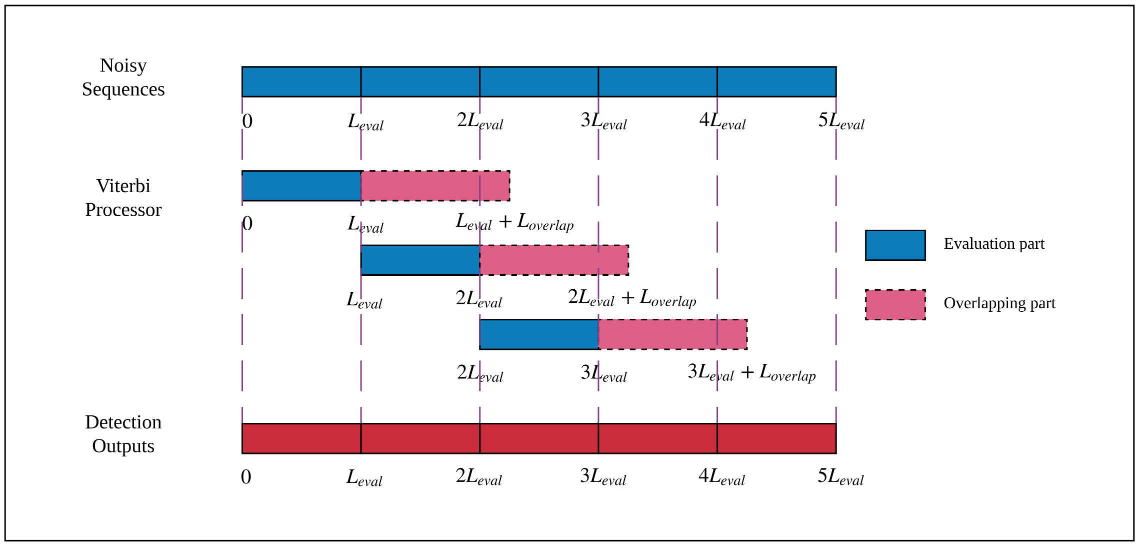

Figure 7.4. Sliding-window evaluation process for Viterbi detection.

that the channel bit spacing $T_c = 1$. The discrete channel model in Fig. 7.2 uses a 41-tap model of the Lorentzian channel $\{g_i\}$ and a 21-tap PR equalizer $\{z_i\}$. The NPML detector is implemented using 4-tap noise predictor, 8-tap noise predictor, and 16-tap noise predictors. In the sliding-window evaluation process, the evaluation length L_{eval} is 10 and the overlapping length $L_{overlap}$ is 20.

7.3 PR-NN: RNN-based Detection

In this section, we present PR-NN (partial response - neural network), an RNN-based detection method for coded partial-response channels. We discuss the details of network architecture, dataset generation, training methodology, evaluation procedure, and computational complexity.

The main idea of PR-NN is to replace the classical detectors with a robust RNN-based detector. The motivation of our approach comes from the GRU-based decoder for noisy convolutional codewords in [110]. Although our project focuses on the application of PR-NN to the coded E^{2}PR4 channel for longitudinal magnetic recording channels, we believe the proposed approach can be easily adapted to other practical magnetic recording systems.

7.3.1 Neural Network Architecture

Thanks to the rapid development of deep learning, many neural network architectures have emerged and shown their power in a variety of application domains. RNNs, in particular, have been adopted in several scenarios involving time-sequential data, making an RNN-based architecture a natural candidate for processing signals produced by a magnetic recording channel with inter-symbol interference and possibly correlated additive noise.

In practice, we exploit more sophisticated recurrent hidden units that implement a gating mechanism [17,19], such as long short-term memory (LSTM) units [44] and gated recurrent units (GRUs) [18]. Comparable performance has been found in networks using GRU and LSTM [19]. Our aim in this work is to explore the potential of RNN-based signal detection in channels with ISI, rather than to compare the performance of different RNN units, so we will only consider networks based on GRU cells.

A GRU schematic is shown in Fig. 7.5. The calculations in a GRU cell can be formulated as follows

$$\begin{aligned}
\boldsymbol{\gamma}_t &= \sigma(\mathbf{W}_{\alpha\gamma} \cdot \boldsymbol{\alpha}_t + \mathbf{b}_{\alpha\gamma} + \mathbf{W}_{h\gamma} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{h\gamma}) \\
\boldsymbol{\mu}_t &= \sigma(\mathbf{W}_{\alpha\mu} \cdot \boldsymbol{\alpha}_t + \mathbf{b}_{\alpha\mu} + \mathbf{W}_{h\mu} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{h\mu}) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_{\alpha\tilde{h}} \cdot \boldsymbol{\alpha}_t + \mathbf{b}_{\alpha\tilde{h}} + \boldsymbol{\gamma}_t \odot (\mathbf{W}_{h\tilde{h}} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{h\tilde{h}})) \\
\mathbf{h}_t &= (\mathbf{1} - \boldsymbol{\mu}_t) \odot \tilde{\mathbf{h}}_t + \boldsymbol{\mu}_t \odot \mathbf{h}_{t-1}
\end{aligned} \tag{7.12}$$

where we use standard notation for weight matrices $\mathbf{W}$, biases $\mathbf{b}$, and activation function σ. In the calculations, the input at time step t is $\boldsymbol{\alpha}_t \in \mathbb{R}^{m_{in}}$, representing m_{in} features. The hidden state at time step $t-1$ is $\mathbf{h}_{t-1} \in \mathbb{R}^{m_h}$. The output at time step t is $\boldsymbol{\beta}_t \in \mathbb{R}^{m_{out}}$. The hidden state at time step t is $\mathbf{h}_t \in \mathbb{R}^{m_h}$. For a GRU cell, the hidden state $\mathbf{h}_t$ is the same as the output $\boldsymbol{\beta}_t$. The reset, update, and new gates are represented by $\boldsymbol{\gamma}_t \in \mathbb{R}^{m_h}$, $\boldsymbol{\mu}_t \in \mathbb{R}^{m_h}$, and $\tilde{\mathbf{h}}_t \in \mathbb{R}^{m_h}$, respectively. The Hadamard product is denoted by $\odot$.

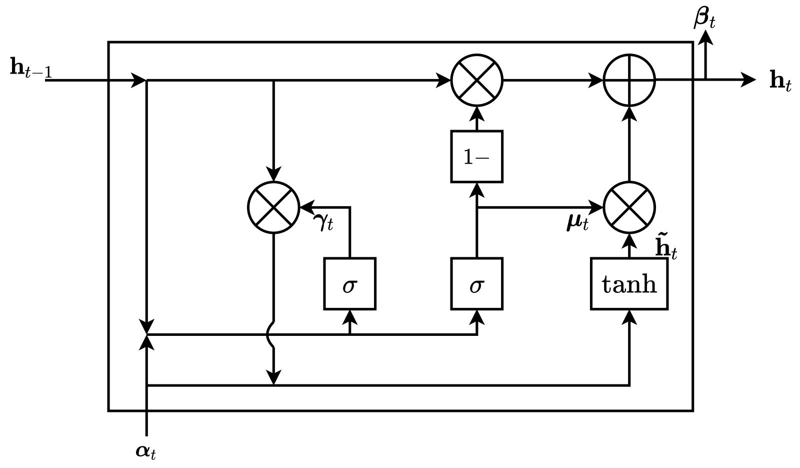

Figure 7.5. Structure of one GRU cell.

We adopt a bi-directional GRU (bi-GRU) architecture [102], which can be understood as two separate GRU networks, one operating in the forward direction and the other operating in the backward direction. As illustrated in Fig. 7.6, in the forward (resp. backward) direction, the forward (resp. backward) GRU component of the bi-GRU at time step t takes the hidden state $\mathbf{h}^f_{t-1}$ from time step $t-1$ (resp. the hidden state $\mathbf{h}^b_{t+1}$ from time step $t+1$) and produces the hidden state $\mathbf{h}^f_t$ for the next GRU cell at time step $t+1$ (resp. the hidden state $\mathbf{h}^b_t$ for the next GRU cell at time step $t-1$). The forward and backward outputs of the bi-GRU cells are concatenated at each time step.

Remark 7. *Our implementation of the GRU will incorporate multi-layer bi-GRU cells as illustrated in Fig. 7.7. We suppose that the total number of time steps in each layer is T_r. The bi-GRU cell operates with simultaneous forward and backward passes at each time step t ($1 \leq t \leq T_r$). The input of the i-th layer ($i \geq 2$) is the hidden state of the previous layer. (We do not dropout any features from the GRU layer outputs.) We set the default initial hidden states of bi-GRU cells (in both directions) to $\mathbf{0}$; specifically, $\mathbf{h}^f_0 = \mathbf{0}$ and $\mathbf{h}^b_{T_r+1} = \mathbf{0}$.*

Referring to the coded E^{2}PR4 state machine in Fig 7.3, we see a conceptual similarity between the forward pass of the bi-GRU and the operation of the Viterbi detector, with [current state/input] and [next state/output] corresponding to [previous hidden state/input] and [next

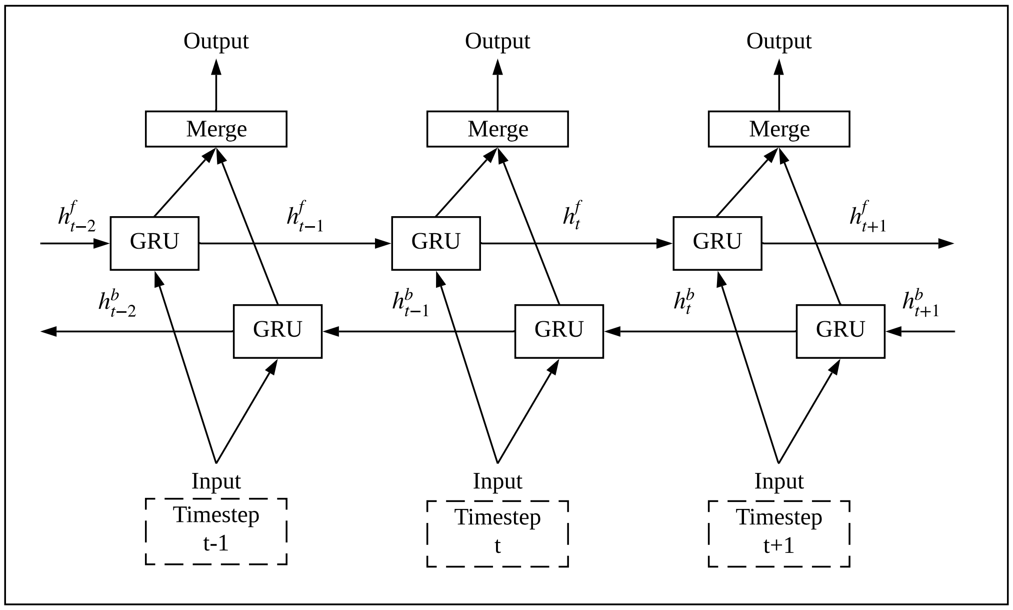

Figure 7.6. Structure of unfolded bi-directional GRUs.

hidden state/output], respectively. Similarly, the forward/backward passes of the bi-GRU bear a conceptual resemblance to the foward/backward passes of the BCJR detector.

In order to design an RNN-based detector for coded PR channels, we will have to train the network with noisy channel output sequences. The number of coded channel output sequences grows exponentially in the length of the channel input (approximately 2^{Rn}, where n is the length of the user input sequence and R is the constrained code rate). This suggests the use of a block-oriented network architecture, with a limited block size. In order for the RNN-based detector to process continuous streaming channel outputs, we adopt a sliding-window approach, similar to the sliding-window implementations of the Viterbi detector and BCJR detector presented in Section 7.2.3, both of which process overlapping blocks.

Referring to Fig. 7.4, the idea is for the bi-GRU network to serve as the detection module for each block of length L_{eval}, using as the network input a length-($L_{eval} + L_{overlap}$) block of the recording channel output. However, when we feed a sequence into the multi-layer bi-GRU cells, the sequence needs to respect the default network initialization conditions in Remark 7, i.e., the initial hidden state for the forward direction and the backward direction should be $\mathbf{0}$.

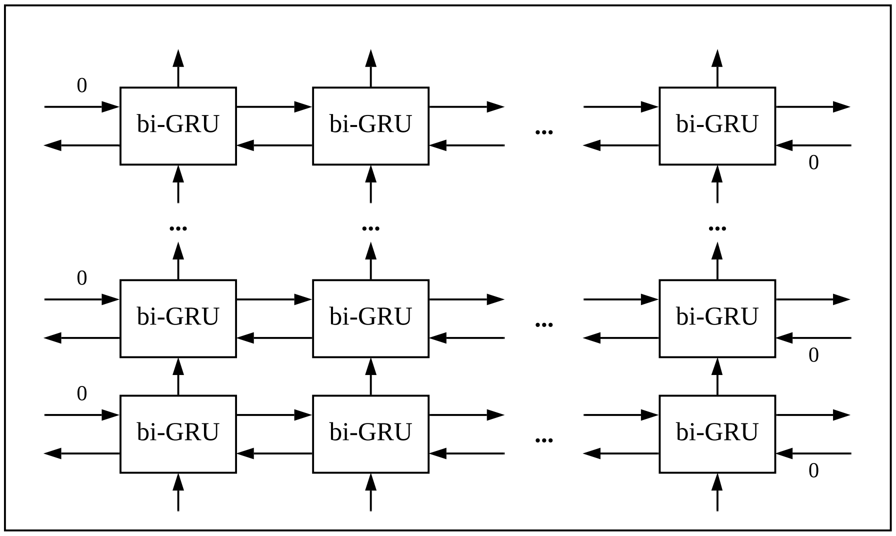

Figure 7.7. Model of multi-layer bi-directional GRUs.

We assume that the default initial hidden state **0** corresponds to the state (0000) in the coded E^{2}PR4 state machine. This poses a problem, because there is no guarantee that the block of inputs to the network correspond to a state sequence in the PR channel state machine that starts and ends in state (0000). To compensate for this, we propose a *zero compensation* approach, in which we append suitable starting and ending dummy values before and after each block to force sequences to start and end at state (0000). The exact rules of the zero compensation approach are provided in the next subsection.

The input sequence to the bi-GRUs is therefore composed of four parts: starting dummy values, evaluation part, overlapping part, and ending dummy values. The respective lengths of these parts are denoted L_{start}, L_{eval}, $L_{overlap}$, and L_{end}. The resulting total number of time steps in the bi-GRUs is $T_r = L_{start} + L_{eval} + L_{overlap} + L_{end}$.

Now we specify the network components of the PR-NN detector. There are three kinds of layers: dense layers $\mathcal{D}(\mathbf{x})$, multi-layer bi-GRU cells $\mathcal{R}(\mathbf{x}, \mathbf{H}_t^f, \mathbf{H}_t^b)$, and the sigmoid layer $\mathcal{S}(\mathbf{x})$. In this network, GRU cells are the key components. For time step t ($1 \leq t \leq T_r$), the details of three kinds of layers are listed below.

1. Dense layer $\mathcal{D}(\mathbf{x})$: given an input vector $\mathbf{x} \in \mathbb{R}^{m_{din}}$, a dense layer defines the following

operation

$$\mathbf{y} = \mathcal{D}(\mathbf{x}) = \mathbf{W}_d \cdot \mathbf{x} + \mathbf{b}_d \tag{7.13}$$

where $\mathbf{y} \in \mathbb{R}^{m_{dout}}$ is the output of the dense layer.

2. Multi-layer bi-GRU cells $\mathcal{R}(\mathbf{x}, \mathbf{H}_t^f, \mathbf{H}_t^b)$: The number of layers is denoted as N_r. The input vector is $\mathbf{x} \in \mathbb{R}^{m_{rin}}$, the forward (resp. backward) hidden state set is $\mathbf{H}_t^f = \{\mathbf{h}_t^{f1}, \mathbf{h}_t^{f2}, \cdots, \mathbf{h}_t^{fN_r}\}$ (resp. $\mathbf{H}_t^b = \{\mathbf{h}_t^{b1}, \mathbf{h}_t^{b2}, \cdots, \mathbf{h}_t^{bN_r}\}$), where $\mathbf{h}_t^* \in \mathbb{R}^{m_{rh}}$ is the hidden state vector and t corresponds to the current time step. The output vector at time step t is $\mathbf{y} \in \mathbb{R}^{m_{rout}}$. The mathematical expression for the network operation is

$$\mathbf{y} = \mathcal{R}(\mathbf{x}, \mathbf{H}_t^f, \mathbf{H}_t^b) \tag{7.14}$$

(The calculations of the GRU cell were presented in (7.12) and the structure of the multi-layer bi-GRU cells was formulated in Remark 7.)

3. Sigmoid layer $\mathcal{S}(\mathbf{x})$: given an input vector $\mathbf{x} \in \mathbb{R}^{m_{sin}}$, the output $\mathbf{y} \in \mathbb{R}^{m_{sout}}$ of the sigmoid layer is

$$\mathbf{y} = \mathcal{S}(\mathbf{x}) : y_i = \frac{1}{1 + e^{x_i}}. \tag{7.15}$$

The PR-NN contains the dense layers (Dense1 layer $\mathcal{D}_1(\mathbf{x})$ and Dense2 layer $\mathcal{D}_2(\mathbf{x})$), the multi-layer bi-GRU cells ($\mathcal{R}(\mathbf{x}, \mathbf{H}_t^f, \mathbf{H}_t^b, t)$), and the Sigmoid layer ($\mathcal{S}(\mathbf{x})$). The input to the PR-NN is derived from a length-T_r noisy channel output sequence, $\mathbf{r} = \{r_1, r_2, \cdots, r_{T_r}\}$, where the total number of time steps in PR-NN is also T_r. To reflect the memory of the E^{2}PR4 channel, we transform the sequence $\mathbf{r}$ into the network input vector $\underline{\mathbf{r}}' = \{\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \cdots, \mathbf{r}^{(T_r)}\}$, where $\mathbf{r}^{(k)} = \{r_{k-4}, r_{k-3}, r_{k-2}, r_{k-1}, r_k\}$.

The network architecture is shown in Fig. 7.8. Taking into account the starting and ending dummy values, we discard the outputs of multi-layer bi-GRUs for the time steps $1 \leq k \leq L_{start}$ and $T_r - L_{end} + 1 \leq k \leq T_r$. The output vector is $\mathbf{y} = \{y_{L_{start}+1}, y_{L_{start}+2}, \cdots, y_{T_r - L_{end}}\} \in$

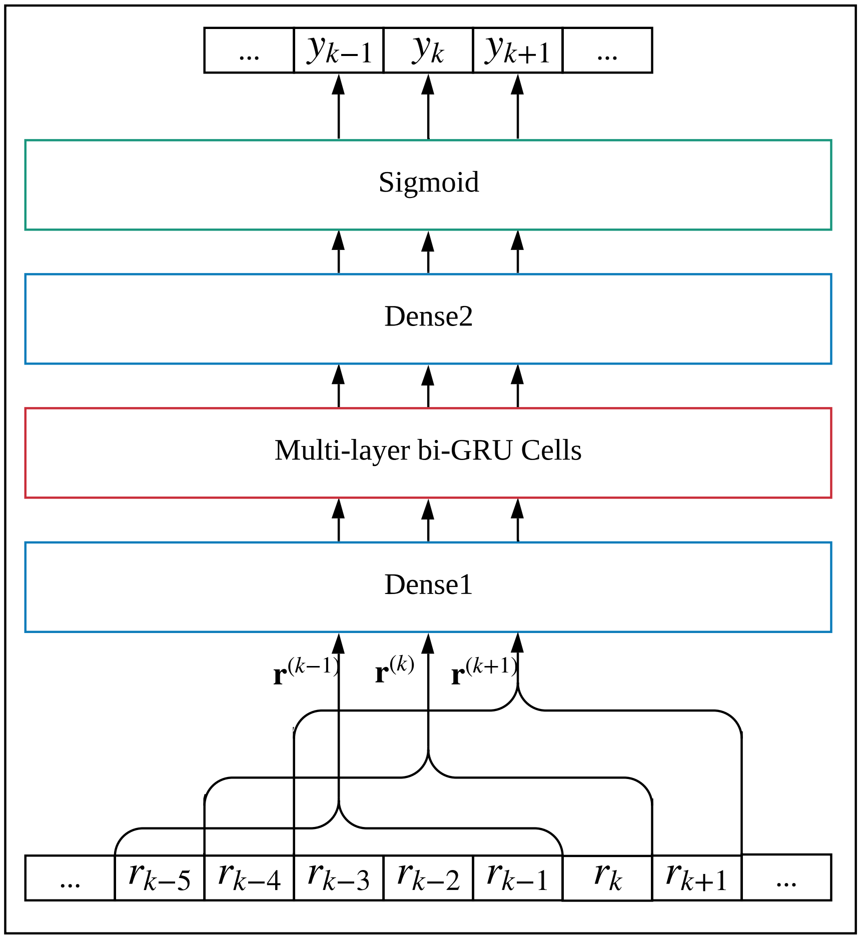

Figure 7.8. Network architecture for proposed PR-NN.

$\mathbb{R}^{(L_{start}+L_{eval})}$ where, for time step $L_{start}+1 \leq k \leq T_r - L_{end}$, the output is given by

$$y_k = \mathcal{S}(\mathcal{D}_2(\mathcal{R}(\mathcal{D}_1(\mathbf{r}^{(k)}), \mathbf{H}_k^f, \mathbf{H}_k^b))). \tag{7.16}$$

Here we use time index k, rather than t, to be consistent with the indexing in the symbol sequences generated by the magnetic recording channel.

Remark 8. *In our experiments, for the network architecture, we chose* $L_{start} = L_{end} = 5$, $L_{eval} = 10$. *The length of the overlapping part is* $L_{overlap} = 20$, *which is the same as the truncation depth in the Viterbi detector. Thereby, the total number of time steps in PR-NN is*

$T_r = 40$. *The parameters* L_{eval} *and* $L_{overlap}$ *are also used in the simulations for the classical detection methods. For Dense1 layer, the number of input features is* 5 *and the number of output features is* 5. *For the multi-layer bi-GRU cells, the number of layers is* $N_r = 4$ *and the number of features in a hidden state is* 50. *For Dense2 layer, the number of input features is* 100 *and the number of output features is* 1.

7.3.2 Data Acquisition

The PR-NN detector recovers PR channel inputs from noisy PR channel outputs. The training dataset of noisy channel outputs is created as follows. First, the coded E^2PR4 channel state machine in Fig. 7.3 is used to generate a length-($L_{eval} + L_{overlap}$) channel input sequence from an arbitrarily chosen initial state, and then suitable distortion is added. In our experiments, we consider three kinds of noise generated from the longitudinal recording system: AWGN, ACN generated by the MMSE PR equalizer for the Lorentzian channel, and total distortion noise n_k generated according to (7.6). In order to train the PR-NN to adapt to different SNRs, the noise in the training set also reflects a range of SNRs.

We use a *zero compensation* rule to ensure the network inputs satisfy the initial settings of bi-GRU cells, which require the corresponding starting and ending states of the PR channel state machine to be (0000). A string of $L_{start} = 5$ starting dummy values is prepended to the noisy channel output sequence according to the initial state, as indicated in Table 7.1. As will be described in Section 7.3.4, PR-NN uses a sliding-window evaluation process, in which the starting state for each truncated input block is determined by the symbols in the previously recovered evaluation block. The starting dummy values are then determined by the zero compensation rule, and since the starting state is assumed to be correct, we do not add noise to the starting dummy values. The final state of the path used to generate the input sequence determines a path to state (0000) and a corresponding string of $L_{end} = 5$ ending dummy values, also shown in Table 7.1. However, since the ending state will be unknown during evalution, we add noise to these ending dummy values in the training sequence to represent noisy outputs corresponding to an unknown

Table 7.1. Starting and Ending Dummy Values for Each State in the Coded E^{2}PR4 State Machine. "Unknown" Means Unknown Starting or Ending State for the Sequence.

State	Starting dummy values	Ending dummy values
(0000)	$\{0,0,0,0,0\}$	$\{0,0,0,0,0\}$
(0001)	$\{0,0,0,0,1\}$	$\{3,2,-2,-3,-1\}$
(0011)	$\{0,0,0,1,3\}$	$\{2,-2,-3,-1,0\}$
(0110)	$\{0,0,1,3,2\}$	$\{-2,-3,-1,0,0\}$
(0111)	$\{0,0,1,3,3\}$	$\{0,-3,-3,-1,0\}$
(1000)	$\{1,3,2,-2,-3\}$	$\{-1,0,0,0,0\}$
(1001)	$\{1,3,2,-2,-2\}$	$\{2,2,-2,-3,-1\}$
(1100)	$\{0,1,3,2,-2\}$	$\{-3,-1,0,0,0\}$
(1110)	$\{0,1,3,3,0\}$	$\{-3,-3,-1,0,0\}$
(1111)	$\{0,1,3,3,1\}$	$\{-1,-3,-3,-1,0\}$
Unknown	$\{0,0,0,0,0\}$	$\{0,0,0,0,0\}$

ending state sequence. The training label for this training sequence is the length-($L_{eval}+L_{overlap}$) channel input sequence.

During evalution, length-($L_{eval}+L_{overlap}$) truncated blocks of a continuous streaming noisy channlel output sequence will be applied to the PR-NN detector. The evaluation labels are the corresponding detected PR-channel input sequences. In order to process the truncated blocks, a string of starting dummy values of length L_{start} is prepended, using the starting state derived from the previously recovered evaluation block and Table 7.1. An all-zero string of length L_{end} is appended to the block, reflecting the fact that the final state of the truncated block is unknown.

Example 1 illustrates the use of Table 7.1 in the generation of dummy values.

Example 4. *If the starting state is* (1001)*, a path which forces the sequence from* (0000) *to* (1001) *is*

$$(0000)\rightarrow(0001)\rightarrow(0011)\rightarrow(0110)\rightarrow(1100)\rightarrow(1001).$$

Thus the corresponding starting dummy values for state (1001) *are* $\{1,3,2,-2,-2\}$*. If the ending state is* (1001)*, a path which drives the sequence from* (1001) *to* (0000) *is*

$$(1001)\rightarrow(0011)\rightarrow(0110)\rightarrow(1100)\rightarrow(1000)\rightarrow(0000).$$

Thus the corresponding ending dummy values for state (1001) *are* $\{2, 2, -2, -3, -1\}$. *During training, noise will then be added to these values.* ■

7.3.3 Training Methodology

The training dataset is fed into the network and we compare the outputs from the network with the training labels. The comparison metric is the loss function. The loss function between an output vector $\mathbf{y}$ and its corresponding channel input $\mathbf{a}$ can be defined as

$$\mathcal{L} = \sum_{k=L_{start}+1}^{L_{start}+L_{overlap}} -a_k \cdot \log(y_k) - (1 - a_k) \cdot \log(1 - y_k) \tag{7.17}$$

To address the complexity of training the nework with the entire channel output space, we adopted the *a-priori ramp-up* training method in [110]. Instead of beginning the training with independent, uniform user data $u_k \sim \text{Bern}(0.5)$, we start training with user data $u_k \sim \text{Bern}(p)$ ($p < 0.5$) and gradually increase p to 0.5. In our case, the training data r_k is generated from the precoded constrained symbols a_k, and a_k is determined by the user data u_k. The biasing probability $p(\text{ep})$ in $u_k \sim \text{Bern}(p)$ is a function of the epoch number ep, defined as

$$p(\text{ep}) = \begin{cases} 0.1 + 0.01 \cdot \lfloor \text{ep}/\#\text{step} \rfloor, & \text{ep} \le 40 \cdot \#\text{step} \\ 0.5, & \text{ep} > 40 \cdot \#\text{step} \end{cases} \tag{7.18}$$

where $\#$step is the number of epochs between increments in the probability. After every $\#$step epochs, the probability $p(\text{ep})$ will increase by 0.01 until 0.5 is reached.

7.3.4 Evaluation Process

The outputs y_k of the network are real values in the range $[0, 1]$. These are converted to binary detector outputs $\hat{a}_k$ using an indicator function: $\hat{a}_k = \mathbb{1}_{\{x>0.5\}}(y_k)$.

The sliding-window concept used in the evaluation process is shown in Fig. 7.9. We assume the state machine has initial state (0000). When the first block of $L_{eval} + L_{overlap}$ symbols

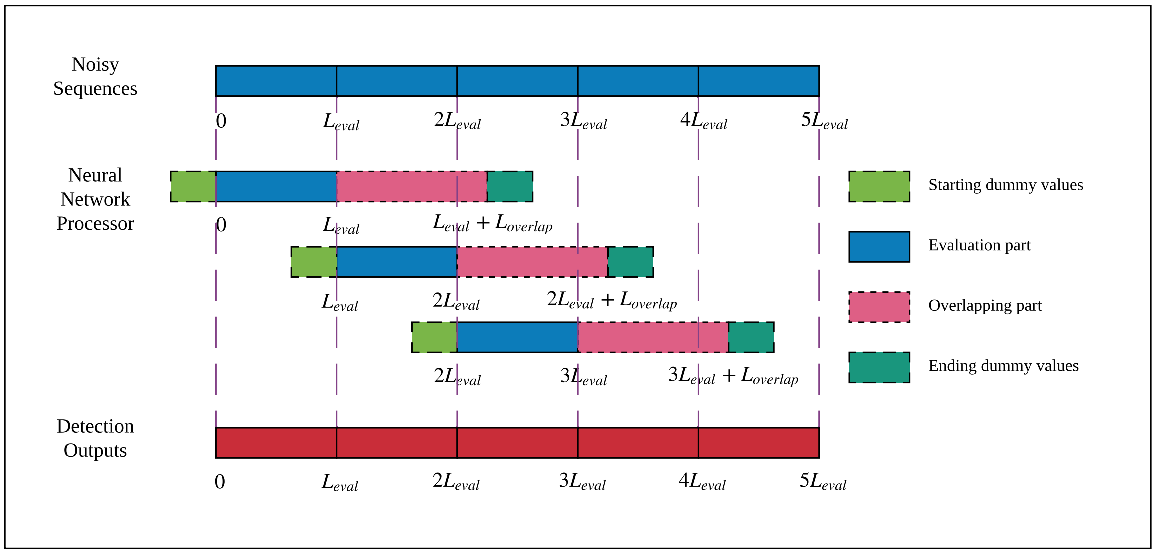

Figure 7.9. Sliding-window evaluation process for PR-NN detector.

are received, the PR - NN detector prepends to the block the starting dummy values corresponding to state (0000) and appends the ending dummy values corresponding to the unknown state. The resulting sequence is processed by the network, producing the detector outputs for the first length-L_{eval} block. The last 4 bits of the recovered block determine the starting state for the next detection stage. When an additional L_{eval} output symbols are received, the sliding window shifts by L_{eval} positions and begins processing the next length-($L_{eval} + L_{overlap}$) truncated block. The starting dummy values for this block depend on the previously recovered starting state, and the ending dummy values correspond to the unknown ending state. The resulting length-L_r block is then processed by the network. This procedure continues until the entire stream of noisy channel outputs is processed.

For a noisy channel output sequence of length L, the evalulation metric is the bit error rate (BER) between the input $\mathbf{a}$ and the detection result $\hat{\mathbf{a}}$, defined as

$$\text{BER} = \frac{1}{L}\sum_{k=1}^{L} \mathbb{1}_{\hat{a}_k \neq a_k}(\hat{a}_k) \tag{7.19}$$

Remark 9. *For the training set, the noise is generated for SNR values (in dB) in the set*

$\mathbb{S} = \{8.5, 9.0, 9.5, 10.0, 10.5\}$ *(The SNR definition can be found in the Appendix.) Throughout the training, Adam optimizer [61] was used with learning rate* 10^{-3}*. The value* $\#$*step used in the a-priori ramp-up training was set to* 50*.*

7.3.5 Computational Complexity Analysis

In this section, we compare the computational complexity of the Viterbi dectector and PR-NN detector. We denote the length of noisy channel output sequence by L and the number of states in the input-constrained channel trellis by N_{st}. Note that $N_{st} \leq 2^{v}$.

We first analyze the asymptotic computational complexity. For a Viterbi detector, the computational complexity can be estimated as $O(N_{st}^2 L) = O(2^{2\nu} L)$.

We now consider the complexity of PR-NN. Noting that the complexity of RNN is asymptotically linear in the number of time steps, we let $\mathcal{T}_{NN}$ denote the computational complexity of one time step in the network. For the evaluation process, the processing of each truncated block of length $L_{eval} + L_{overlap}$ requires $(L_{start} + L_{eval} + L_{overlap} + L_{end})$ steps, where $L_{start} = L_{end} = L_{dummy}$ and $L_{overlap}$ has the form $A\nu$, for some constants L_{dummy} and A. Thus, the associated complexity is nominally $C = (2L_{dummy} + L_{eval} + A\nu)\mathcal{T}_{NN}$. Since the multi-layer bi-GRU outputs corresponding to dummy values can be ignored by the Dense2 layer, the complexity is actually $C' \leq C$. Processing the entire network input stream involves approximately L/L_{eval} blocks, so the overall complexity is approximately $LC'/L_{eval} = O(\nu L)$.

Thus, in the asymptotic view as the stream length L becomes large, the complexity of PR-NN compares favorably to that of Viterbi detection in PRML. Analogous complexity analysis for PRMAP and NPML detection leads to similar conclusions.

To more precisely evaluate the constants in the “order of” complexity estimate, we analyze in more detail the complexity of operations in the Viterbi and PR-NN detectors. For a Viterbi detector, a typical implementation consists of a pipeline of three units: a branch metric unit (BMU), an add-compare-select unit (ACSU), and a survivor-memory-unit (SMU) [27]. The complexity of the add-compare-select unit for the PR Viterbi decoder with N_{st} states is approximately N_{st} adders, where $N_{st}/2$ adders are required for the addition of branch metrics and the other $N_{st}/2$ make up the complexity of the $N_{st}/2$ two-level compares [28]. If we denote the complexity of this pipeline at each time step as $\mathcal{T}_0$, the overall complexity can be estimated as $\mathcal{T}_0 L$. The MAP decoder can be implemented with no more than 4 times that of a Viterbi decoder [116].

We now discuss the complexity of PR-NN. The number of multiplication operations and the number of addition operations in a combined matrix-vector product and vector addition operation such as $\mathbf{W} \cdot \boldsymbol{x} + \mathrm{b}$ are both equal to the number of elements in the matrix. We let $\mathcal{T}_1$ denote the complexity of a pair of multiplication and addition operations, and assume that the complexity of the activation and indicator functions can be ignored. Denoting the numbers of weights in Dense1 layer, multi-layer bi-GRU cells (one time step) and Dense2 layer as n_{D1}, n_G and n_{D2}, respectively, we see that $\mathcal{T}_{NN} = (n_{D1} + n_G + n_{D2})\mathcal{T}_1$. Because the dummy parts are ignored by Dense2 layer, the complexity is actually $C' = C - 2L_{dummy} n_{D2} \mathcal{T}_1$. Taking into account the sliding-window process, the overall complexity of PR-NN is LC'/L_{eval}.

The following example uses the E^{2}PR4 channel to illustrate the calculation of the operational complexity of Viterbi and PR-NN detectors.

Example 5. *For the coded E^{2}PRML detector, the trellis at each time step has* 10 *states in Fig. 7.3. At each time step, the BMU will compute* 16 *branch metrics, the ACSU will compute* 10 *path metrics, and the SMU will store the decisions made by the ACSU. The complexity of the add-compare-select operation is about* 10 *adders.*

For the PR-NN, based on the parameters in Remark. 8, the number of weights (including biases) in the network layers are $n_{D1} = 30$, $n_G = 153.9k$ *and* $n_{D2} = 101$. *Thus,* $C' = 40\mathcal{T}_{NN} - 1010\mathcal{T}_1$. *The overall complexity of PR-NN is approximately* $615k \cdot \mathcal{T}_1 L$.

7.4 Experimental Results

In this section, we present our experimental results for PR-NN detection of the coded E^{2}PR4 channel. We consider three scenarios. First, we train the network separately on ideal

PR signals with AWGN or ACN generated by a PR equalizer. Then we use joint training on different combinations of AWGN and ACN, as well as on a combination of ACN corresponding to different channel densities. Finally, we train the network with "realistic" equalized Lorentzian channel signals and distortions that include colored noise and misequalization errors. Together, these experiments shed light on the robustness of PR-NN detection. We note that under each scenario, with the a-priori ramp-up approach, PR-NN training converges after $(40 \cdot \#\text{step})$ epochs based on monitoring the loss function.

7.4.1 Experimental Setup

In the first scenario, we train PR-NN with only one kind of noise, i.e., AWGN or ACN. For ACN, the colored noise is generated by applying the MMSE PR equalizer to AWGN samples. According to (7.4), the ACN is affected by the channel density parameter PW_{50}/T_c, so we use two different, representative values of PW_{50}/T_c in our experiments. In the training process, the batch size for each SNR in $\mathbb{S}$ is 30.

In the second scenario, we assess PR-NN robustness by training a single network to adapt to two different types of noise: (a) AWGN and ACN at $PW_{50}/T_c = 2.54$, or (b) ACN at $PW_{50}/T_c = 2.54$ and at $PW_{50}/T_c = 2.88$. The training batch size settings for case (a) are shown in lines 1, 2, and 3 and for case (b) in line 4 of Table 7.2.

In the third scenario, the channel outputs represent a more realistic, MMSE-equalized Lorentzian channel with misequalization errors and ACN. To assess robustness to different channel densities, we train the PR-NN with separate datasets at $PW_{50}/T_c = 2.54$ or $PW_{50}/T_c = 2.88$, and then jointly with a dataset combining the two densities. Batch sizes are shown in lines 5, 6, and 7 of Table 7.2.

7.4.2 Scenario 1: Individual Training Experiments

In scenario 1, PR-NN is only trained with one noise, i.e., AWGN, ACN ($PW_{50}/T_c = 2.54$), ACN($PW_{50}/T_c = 2.88$), where the batch size is 30 for each SNR in $\mathbb{S}$. The evaluation

Table 7.2. Batch Size Settings for the Training Datasets and Evaluation Cases in Each Experiment of the Three Scenarios.

SNR	8.5dB	9.0dB	9.5dB	10.0dB	10.5dB	8.5dB	9.0dB	9.5dB	10.0dB	10.5dB
Noise type	White noise					Colored noise ($PW_{50}/T_c = 2.54$)				
Experiment 2.1	30	30	30	30	30	30	30	30	30	30
Experiment 2.2	40	40	40	40	40	20	20	20	20	20
Experiment 2.3	50	50	50	50	50	10	10	10	10	10
Noise type	Colored noise ($PW_{50}/T_c = 2.54$)					Colored noise ($PW_{50}/T_c = 2.88$)				
Experiment 2.4	30	30	30	30	30	30	30	30	30	30
Noise type	"Realistic" system ($PW_{50}/T_c = 2.54$)					"Realistic" system ($PW_{50}/T_c = 2.88$)				
Experiment 3.1	30	30	30	30	30	-	-	-	-	-
Experiment 3.2	-	-	-	-	-	30	30	30	30	30
Experiment 3.3	30	30	30	30	30	30	30	30	30	30

results over the corresponding channels are described below.

Experiment 1.1: We trained the PR-NN with AWGN. As shown in the BER plot in Fig. 7.10, coded E^{2}PRML achieves the expected 2.2dB gain over uncoded E^{2}PRML with Viterbi detection [6]. In both cases, E^{2}PRMAP, implemented by *max-log-map* approximation, performs essentially the same as E^{2}PRML. (In the subsequent experiments, because E^{2}PRMAP shows similar performance with E^{2}PRML, we only present the simulation results of E^{2}PRML.) The user data BER of the coded E^{2}PRML channel suffers a loss of about 0.9dB due to error propagation of the sliding-block decoder.

We see that the PR-NN achieves performance within 0.1dB of the optimal Viterbi detector. There is a similar gap in performance for the user data BER. (In subsequent experiments, we present only the BER results at the detector output, not the user data results.)

At all SNRs, the histograms of error positions observed within an evaluation block of length $L_{eval} = 10$ for the PR-NN detector and the Viterbi detector are approximately uniform, indicating that the overlapping part is providing "reliable" state information for the evaluation part.

Experiment 1.2: The PR-NN detector is trained and evaluated with ACN ($PW_{50}/T_c = 2.54$). Referring to Fig. 7.11, we see that the performance of coded E^{2}PRML is degraded in colored noise. The NPML detectors with 4-tap, 8-tap, and 16-tap predictors realize gains of

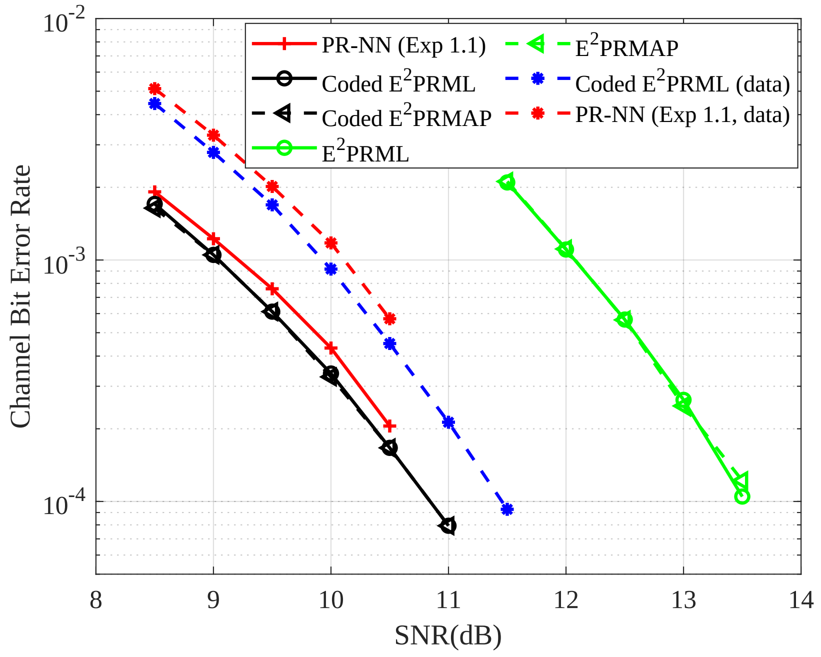

Figure 7.10. Scenario 1: Individual training with AWGN.

0.4dB, 0.5dB, and 0.6dB gain, respectively, over coded E^{2}PRML. Note that for these experiments, the NPML detector designs assume no misequalization error. The PR-NN detector has very similar performance to the 8-tap NPML detector.

Experiment 1.3: The PR-NN detector is trained and tested with ACN ($PW_{50}/T_c = 2.88$). Referring to Fig. 7.12, we see that, in this case, the NPML detectors with 4-tap, 8-tap, and 16-tap predictors realize gains of 1.3dB, 1.5dB, and 1.55dB, respectively, over coded E^{2}PRML. The PR-NN performance is very close to that of the 16-tap NPML detector. In fact, at SNR=10.5dB, the BER achieved by PR-NN is even slightly better.

7.4.3 Scenario 2: Joint Training Experiments

In scenario 2, we train a single PR-NN using a dataset that combines noisy outputs representing different recording channels, and then evaluate its performance on both channels. Four different situations are considered.

Experiments 2.1, 2.2, 2.3: First, the PR-NN detector is trained with both AWGN and

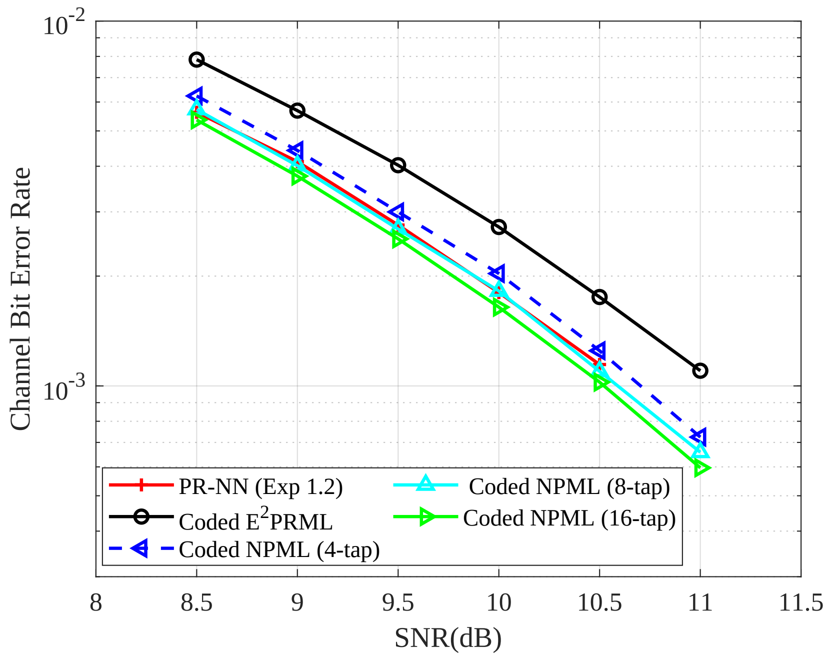

Figure 7.11. Scenario 1: Individual training with ACN (for two channel densities) when $PW_{50}/T_c = 2.54$.

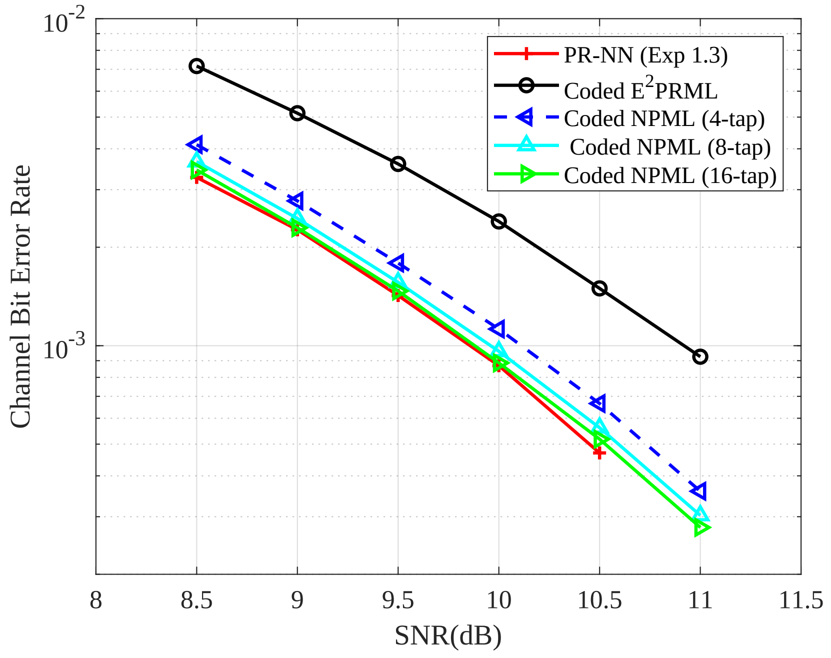

Figure 7.12. Scenario 1: Individual training with ACN (for two channel densities) when $PW_{50}/T_c = 2.88$.

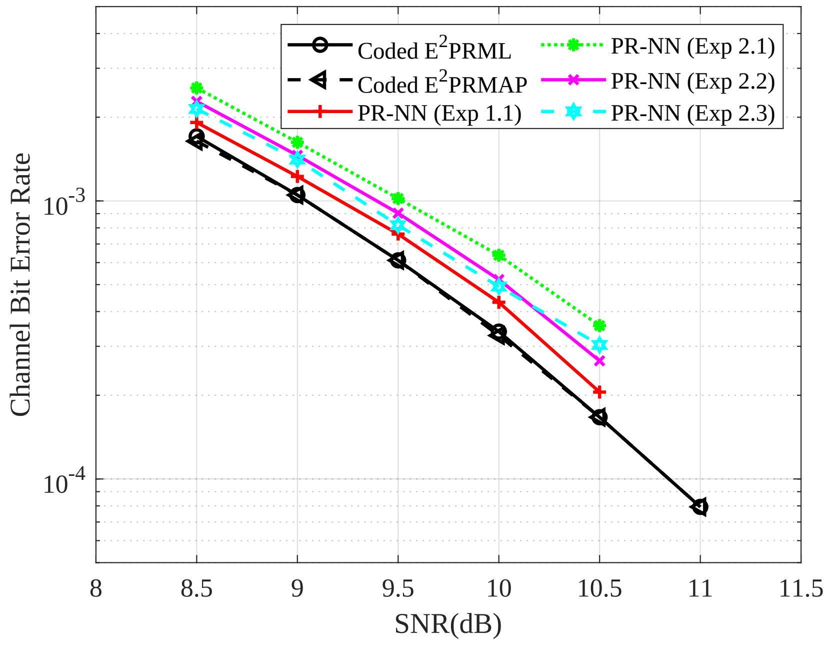

Figure 7.13. Scenario 2: Joint training with AWGN and ACN ($PW_{50}/T_c = 2.54$) and evaluation on AWGN.

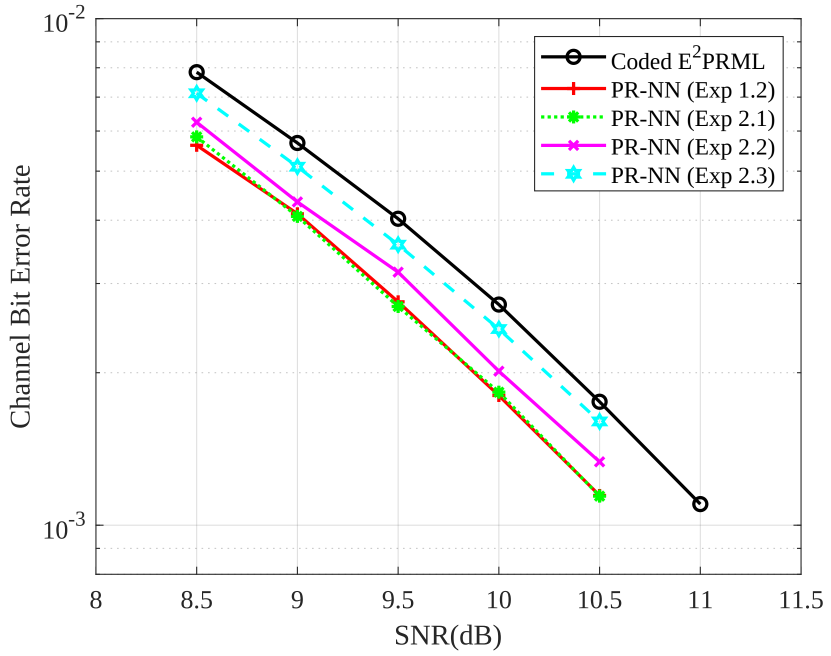

Figure 7.14. Scenario 2: Joint training with AWGN and ACN ($PW_{50}/T_c = 2.54$) and evaluation on ACN ($PW_{50}/T_c = 2.54$).

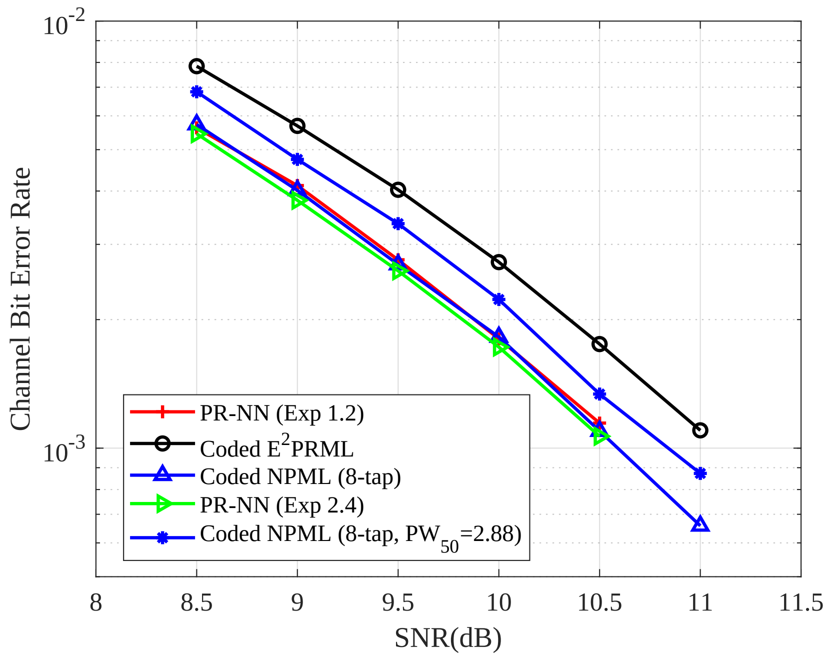

Figure 7.15. Scenario 2: Joint training with ACN when $PW_{50}/T_c = 2.54$ and ACN when $PW_{50}/T_c = 2.88$ and evaluation on ACN when $PW_{50}/T_c = 2.54$.

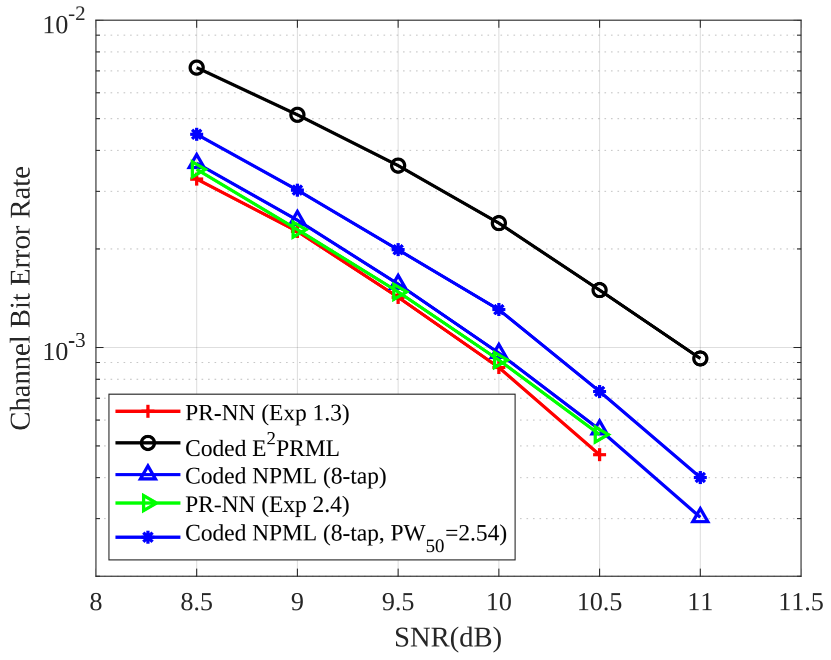

Figure 7.16. Scenario 2: Joint training with ACN when $PW_{50}/T_c = 2.54$ and ACN when $PW_{50}/T_c = 2.88$ and evaluation on ACN when $PW_{50}/T_c = 2.88$.

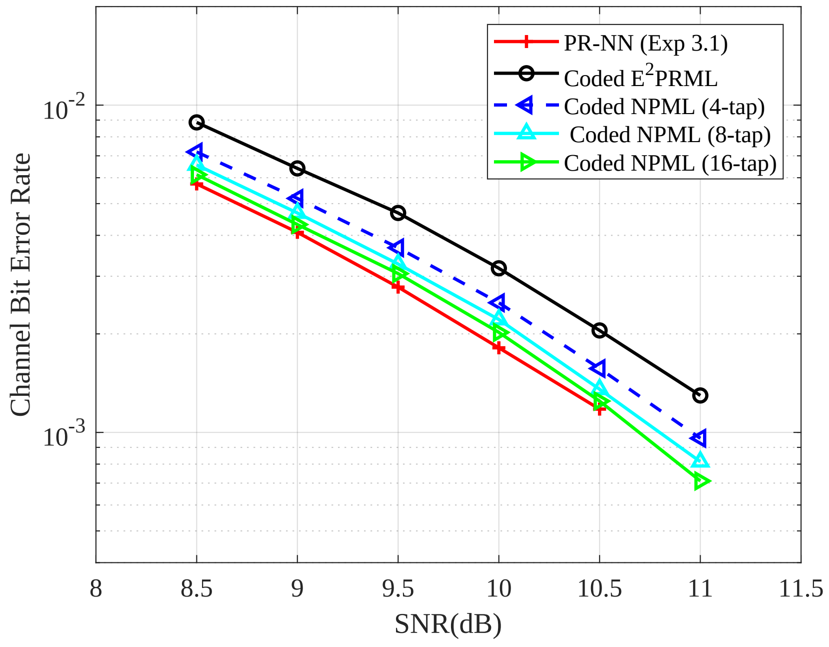

Figure 7.17. Scenario 3: Individual training with "realistic" datasets at $PW_{50}/T_c = 2.54$ and evaluation on "realistic" datasets at $PW_{50}/T_c = 2.54$.

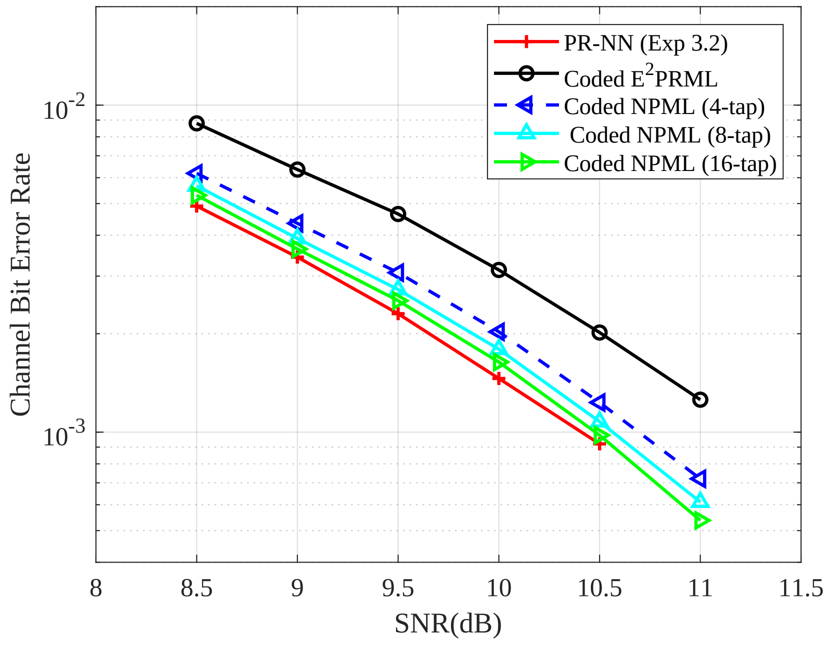

Figure 7.18. Scenario 3: Individual training with "realistic" datasets at $PW_{50}/T_c = 2.88$ and evaluation on "realistic" datasets at $PW_{50}/T_c = 2.88$.

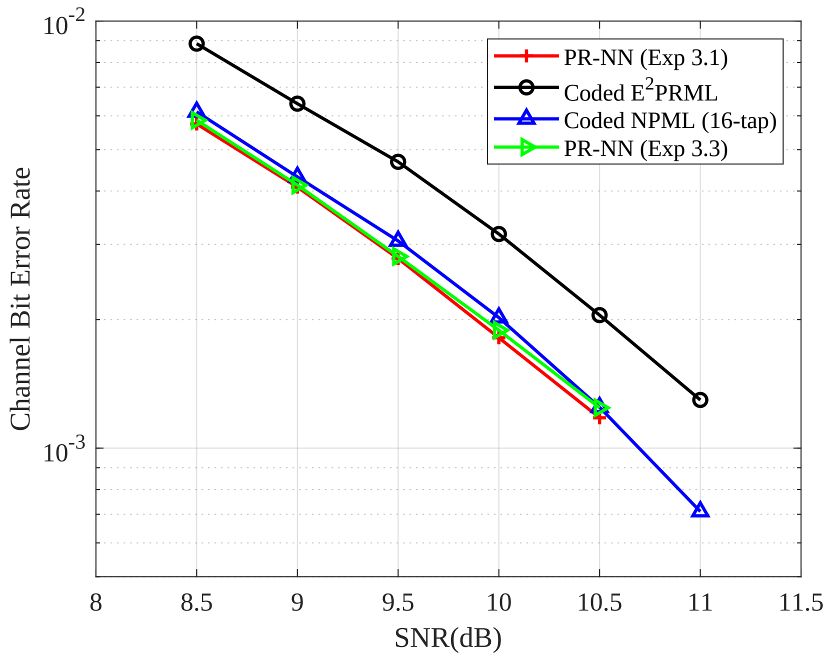

Figure 7.19. Scenario 3: Joint training with "realistic" datasets for both $PW_{50}/T_c = 2.54$ and $PW_{50}/T_c = 2.88$ and evaluation on "realistic" datasets for $PW_{50}/T_c = 2.54$.

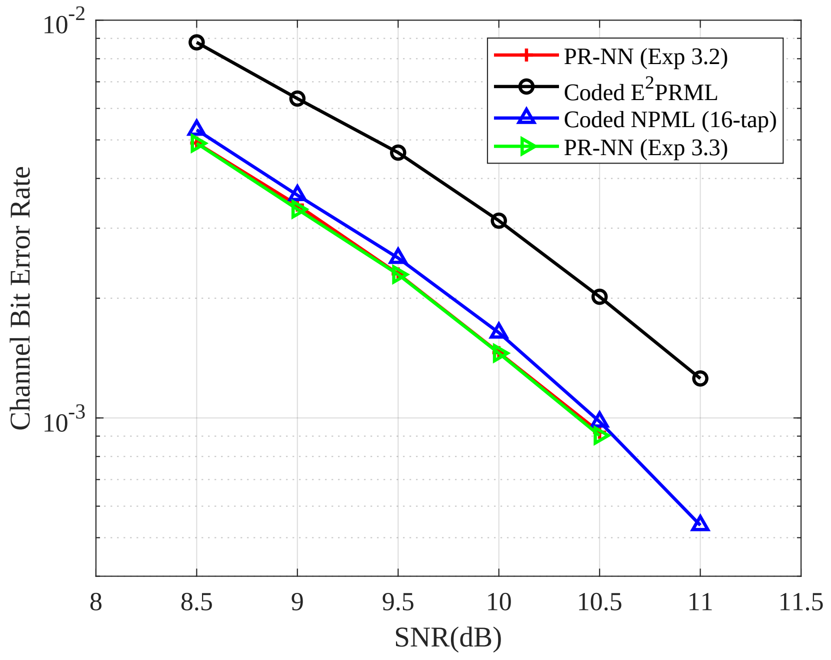

Figure 7.20. Scenario 3: Joint training with "realistic" datasets for both $PW_{50}/T_c = 2.54$ and $PW_{50}/T_c = 2.88$ and evaluation on "realistic" datasets for $PW_{50}/T_c = 2.88$.

ACN ($PW_{50}/T_c = 2.54$). Experiments 2.1, 2.2, and 2.3 use different relative batch sizes for AWGN and ACN samples within the training set, as shown in Table 7.2. The simulation results are summarized in Fig. 7.13 and Fig. 7.14. The solid red curve in Fig. 7.13 represents the best performance in AWGN achieved by the network individually trained with AWGN (Experiment 1.1). When trained with the combined datasets, the jointly trained PR-NNs suffer losses of 0.4dB, 0.2dB, and 0.1dB, respectively, with respect to the network individually trained with ACN (Experiment 1.2), with the larger relative batch size for AWGN giving the best performance. On the other hand, as shown in Fig. 7.14, when we evaluate the jointly-trained PR-NN under ACN ($PW_{50}/T_c = 2.54$), the performance losses are 0dB, 0.1dB, and 0.4dB, respectively, with increasing relative AWGN batch size.

These results suggest that the best compromise in terms of robustness of performance is offered by the jointly trained PR-NN in Experiment 2.2, with losses of only 0.2dB in AWGN and 0.1dB in ACN.

Experiment 2.4: In this experiment, we explore the robustness of a jointly-trained PR-NN detector at different channel densities ($PW_{50}/T_c = 2.54$ and $PW_{50}/T_c = 2.88$). The training batch sizes are shown in Table 7.2. The simulation results in Fig. 7.15 show that the resulting PR-NN detector matches the performance of the individually-trained network in Experiment 1.2. Moreover, as seen in Fig. 7.16, the performance is only slightly worse than that of the network trained in Experiment 1.3. Thus, the jointly-trained network appears to offer adaptivity to different recording densities, which can arise from system variations in temperature and head flying height, or at different disk radii.

Interestingly, the NPML detectors do not exhibit the same sort of robustness as the PR-NN detector. Fig. 7.15 shows that the NPML detector optimized for $PW_{50}/T_c = 2.88$ experiences a performance loss of 0.2dB with respect to the NPML detector properly optimized for $PW_{50}/T_c = 2.54$. Similarly, we see in Fig. 7.16 that the NPML detector designed for $PW_{50}/T_c = 2.54$ incurs a penalty of 0.3dB compared to the NPML detector designed for $PW_{50}/T_c = 2.88$.

7.4.4 Scenario 3: “Realistic” Equalized Lorentzian Channel

In a “realistic” recording system modeled as an equalized Lorentzian channel, the signal distortions include both colored noise and misequalization errors, as shown in (7.6). In this third set of experiments, we first compare the performance of NPML detection and PR-NN detection for such a system at two channel densities. We then assess the robustness of a PR-NN detector trained jointly for use at both densities. Note that in these experiments, the NPML detector designs take into account both colored noise and misequalization errors.

Experiment 3.1: The simulation results for a PR-NN detectors trained individually at $PW_{50}/T_c = 2.54$ are shown in Fig. 7.17. The NPML detectors with 4-tap, 8-tap, and 16-tap predictors have gains of 0.3dB, 0.4dB and 0.45dB, respectively, over coded E^{2}PRML. The PR-NN detector trained with the “realistic” channel dataset achieves even slightly better performance than the 16-tap NPML detector.

Experiment 3.2: In Fig 7.18, we consider the channel with density $PW_{50} = 2.88$. Here the gains of the NPML detectors with 4-tap, 8-tap, and 16-tap predictors are 0.5dB, 0.6dB, and 0.7dB, respectively, over coded E^{2}PRML. The PR-NN detector trained with the corresponding dataset surpasses that of the 16-tap NPML detector.

Experiment 3.3: As in Experiment 2.4, we explore the adaptability of PR-NN detection to changes in recording density. The results obtained after training with a combined dataset of “realistic” equalized Lorentzian channel outputs for $PW_{50} = 2.54$ and $PW_{50} = 2.88$ are shown in Fig. 7.17 and Fig. 7.18. In Fig. 7.17, corresponding to $PW_{50} = 2.54$, we see that the jointly-trained network essentially matches the performance of the individually trained network (Experiment 3.1), surpassing the 16-tap NPML detector. Similarly, Fig. 7.18 shows that the jointly-trained PR-NN detector preserves the performance of the network individually trained at $PW_{50} = 2.88$ (Experiment 3.2). The robustness of PR-NN detection in this more realistic channel setting is thus confirmed.

7.4.5 Experimental Analysis

Our results from Scenario 1 demonstrate that the PR-NN detection architecture can achieve performance close to Viterbi detection and NPML detection on coded E^2PR4 channels in AWGN and ACN, respectively, over a range of SNRs. In Scenario 2, we saw that a PR-NN detector jointly trained for AWGN and ACN shows greater tolerance to ACN than the Viterbi detector, and retains comparable performance in AWGN. When jointly trained in ACN corresponding to equalizers for two different channel densities, the PR-NN detector again exhibits robust performance over a range of SNRs. Finally, when evaluated on a more realistic equalized Lorentzian channel model with both ACN and misequalization errors, the PR-NN detectors designed individually for two channel densities surpass the performance of 16-tap NPML detectors over a range of SNRs. The jointly-trained PR-NN detector maintains the performance of the individually-trained networks at both densities, displaying a robustness that the NPML detectors fail to offer.

The near-optimal performance and robustness of PR-NN can be explained as follows: 1) the RNN-based structure of PR-NN, which exploits the time-sequential connections between cells, reflects the nature of the signal generated by the ISI channel; 2) non-linear functions included in the RNN help PR-NN to model the effects of a variety of noise sources and distortions; and 3) the sliding-window evaluation process helps the block-wise PR-NN architecture to detect the noisy outputs in streaming fashion, in analogy to the classical detection methods used in magnetic recording channels.

7.5 Conclusion

In this paper, we first formulated a magnetic recording channel model and reviewed three classical detectors. Then, we proposed PR-NN, an RNN-based detection approach for coded partial-response channel models. The PR-NN detector processes the noisy outputs of the equalized recording channel in a block-streaming fashion, with computational complexity comparable to that of classical sequence detectors. Simulation results confirm the attractive performance of PR-NN when compared to classical detection algorithms and, moreover, demonstrate a robustness to different noise characteristics and channel densities that classical methods can not provide.

7.6 Appendix

We consider three noise scenarios in our experiments: AWGN, ACN and a "realistic" system with equalized noise and misequalization error. The general expression for the noise is given in (7.6). The signal-to-noise (SNR) ratio is defined in each scenario as follows:

1. AWGN: An ideal channel is assumed and the additive noise term n_k is simply AWGN. The SNR is defined as $SNR_1 = 10\log_{10}(E/\sigma^2)$, where σ^2 is the variance of the Gaussian random variable n_k. Here E is a constant related to the output-voltage amplitude in the recording channel. According to [56], we utilize the matched-filter bound (MFB) as E, and we define MFB as the the distance

$$d_{MFB}^2 = ||x(D)||^2. \tag{7.20}$$

   In the E^{2}PR4 channel, $d_{MFB}^2 = 10$.

2. ACN: Misequalization error is ignored and the noise term is $n_k = \sum_i z_i \eta_{k-i}$, where $\{z_i\}$ represent the normalized coefficients of the PR equalizer and η_k is AWGN. The SNR is defined as $SNR_2 = 10\log_{10}(E/\sigma^2)$, where E is the MFB and σ^2 is the variance of η_k.

3. "Realistic" system: There is misequalization error and colored noise due to equalization. The noise term n_k is given by (7.6). The SNR is defined as $SNR_3 = 10\log_{10}(E/\sigma^2)$, where E is the MFB and σ^2 is the variance of η_k.

Acknowledgement

This chapter is in part a reprint of the material in the paper [128]: Simeng Zheng, Yi Liu, and Paul H. Siegel, “PR-NN: RNN-based detection for coded partial-response channels,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 1967–1982, July 2021. The work was also presented in part at ITW 2020 [129]: Simeng Zheng, Yi Liu, and Paul H. Siegel, “RNN-based detection for coded partial-response channels,” in *Proc. IEEE Information Theory Workshop (ITW)*, Riva del Garda, Italy, Apr. 2021. The author would like to thank the collaborators in these publications.

Bibliography

[1] R. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes–An application of symbolic dynamics to information theory," *IEEE Trans. Inf. Theory*, vol. 29, no. 1, pp. 5–22, Jan. 1983.

[2] M. Asadi, X. Huang, A. Kavcic, and N. P. Santhanam, "Optimal detector for multilevel NAND flash memory channels with intercell interference," *IEEE J. Select. Areas Commun.*, vol. 32, no. 5, pp. 825–835, May 2014.

[3] C. A. Aslam, Y. L. Guan, and K. Cai, "Detector for MLC NAND flash memory using neighbor-a-priori information," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* vol. 24, no. 9, pp. 2827–2836, Sep. 2016.

[4] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[5] A. Behboodi, S. Zheng, J. B. Soriaga, M. Welling, and T. Orekondy, "Data-driven probabilistic modeling of wireless channels using conditional variational auto-encoders," US Patent 0123966, Apr. 2022.

[6] R. T. Behrens and A. J. Armstrong, "An advanced read/write channel for magnetic disk storage," in *Proc. 26th Asdomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Oct. 1992, pp. 956–960.

[7] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes - A syndrome-based approach," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Vail, CO, USA, June. 2018.

[8] Z. T. Blair, "Characterization of TLC flash memory," Master's Thesis, University of California San Diego, 2017.

[9] V. Braun and K. A. S. Immink, "An enumerative coding technique for DC-free runlength-limited sequences," *IEEE Trans. Commun.*, vol. 48, no. 12, pp. 2024–2031, Dec. 2000.

[10] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. of the IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.

[11] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling,” in *Proc. Design, Automation & Test in Europe Conf. & Exhib.*, Grenoble, France, Mar. 2013.

[12] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis,” in *Proc. Design, Automation & Test in Europe Conf. & Exhib.*, Dresden, Germany, Mar. 2012.

[13] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, “Data retention in MLC NAND flash memory: Characterization, optimization, and recovery,” in *IEEE Int. Symp. High Perform. Comput. Architectures (HPCA)*, Burlingame, CA, USA, Mar. 2015.

[14] J. Centers, X. Tan, A. Hareedy, and R. Calderbank, “Power spectra of constrained codes with level-based signaling: Overcoming finite-length challenges,” *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 4971–4986, Aug. 2021.

[15] Y. M. Chee, J. Chrisnata, H. M. Kiah, S. Ling, T. T. Nguyen, and V. K. Vu, “Capacity-achieving codes that mitigate intercell interference and charge leakage in Flash memories,” *IEEE Trans. Inf. Theory*, vol. 65, no. 6, pp. 3702–3712, Jun. 2019.

[16] P. Chen, K. Cai and S. Zheng, “Rate-adaptive protograph LDPC codes for multi-level-cell NAND flash memory," *IEEE Commun. Lett.*, vol. 22, no. 6, pp. 1112–1115, Jun. 2018.

[17] K. Cho, B. V. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, Oct. 2014.

[18] K. Cho, B. V. Merrienboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” in *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, Oct. 2014.

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” in *Proc. Neural Inf. Process. Syst. (NIPS) 2014 Workshop on Deep Learning*, Dec. 2014.

[20] R. D. Cideciyan, F. Dolivo, R. Hermann, W. Hirt, and W. Schott, “A PRML system for digital magnetic recording,” *IEEE J. Select. Areas Commun.*, vol. 10, pp. 38–56, Jan. 1992.

[21] J. D. Coker, E. Eleftheriou, R. L. Galbraith, and W. Hirt, “Noise-predictive maximum likelihood (NPML) detection,” *IEEE Trans. Magn.*, vol. 34, no. 1, pp.110-117, Jan. 1998.

[22] T. Cover, “Enumerative source encoding,” *IEEE Trans. Inf. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[23] B. Dabak, A. Hareedy, and R. Calderbank, “Non-binary constrained codes for two-dimensional magnetic recording,” *IEEE Trans. Magn.*, vol. 56, no. 11, pp. 1–10, Nov. 2020.

[24] J. Deng, W. Dong, R. Socher, L-J. Li, K. Li, and F-F. Li, “Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, FL, USA, June, 2009.

[25] G. Dong, S. Li, and T. Zhang, “Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory,” *IEEE Trans. Circuits Syst. I, Reg. Papers,* vol. 57, no. 10, pp. 2718–2728, Oct. 2010.

[26] N. Farsad and A. J. Goldsmith, “Neural network detection of data sequences in communication systems,” *IEEE Trans. Signal Process.*, vol. 66, pp. 5663-5678, Nov. 2018.

[27] G. P. Fettweis and H. Meyr, “High-speed parallel Viterbi decoding: algorithm and VLSI-architecture,” *IEEE Commun. Magazine*, vol.29, no.5, pp. 46–55, 1991.

[28] G. P. Fettweis, R. Karabed, P. H. Siegel, and H. K. Thapar, “Method and means for detecting partial response waveforms using a modified dynamic programming heuristic,” U.S. Patent 5 430 744, Jul. 1995.

[29] G. D. Forney Jr, “Convolutional codes II. Maximum-likelihood decoding,” *Information and Control*, vol.25, no.3, pp. 222–266, 1974.

[30] P. A. Franaszek, “Sequence-state methods for run-length-limited coding,” *IBM J. Res. Dev.*, vol. 14, no. 4, pp. 376–383, Jul. 1970.

[31] I. J. Goodfellow, J. P.-Abadie, M. Mirza, B. Xu, D. W.-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Proc. Neural Inf. Process. Syst. (NIPS)*, Montréal, Canada, Dec. 2014, pp. 2672–2680.

[32] T. Gruber, S. Cammerer, J. Hoydis, and S. T. Brink, “On deep learning-based channel decoding,” in *Proc. IEEE 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Baltimore, MD, USA, Mar. 2017, pp. 1–6.

[33] A. Hareedy and R. Calderbank, “Asymmetric LOCO codes: Constrained codes for Flash memories,” in *Proc. 57th Annual Allerton Conf. Commun., Control, and Computing*, Monticello, IL, USA, Sep. 2019, pp. 124–131.

[34] A. Hareedy and R. Calderbank, “LOCO codes: Lexicographically-ordered constrained codes,” *IEEE Trans. Inf. Theory*, vol. 66, no. 6, pp. 3572–3589, Jun. 2020.

[35] A. Hareedy, B. Dabak, and R. Calderbank, “Managing device lifecycle: Reconfigurable constrained codes for M/T/Q/P-LC Flash memories,” *IEEE Trans. Inf. Theory*, vol. 67, no. 1, pp. 282–295, Jan. 2021.

[36] A. Hareedy, R. Kuditipudi, and R. Calderbank, “Minimizing the number of detrimental objects in multi-dimensional graph-based codes,” *IEEE Trans. Commun.*, vol. 68, no. 9, pp. 5299–5312, Sep. 2020.

[37] A. Hareedy, B. Dabak, and R. Calderbank, “The secret arithmetic of patterns: A general method for designing constrained codes based on lexicographic indexing,” *IEEE Trans. Inf. Theory*, vol. 68, no. 9, pp. 5747–5778, Sep. 2022.

[38] A. Hareedy, S. Zheng, P. H. Siegel, and R. Calderbank, “Efficient constrained codes that enable page separation in modern flash memories,” *IEEE Trans. Commun.*, vol. 71, no. 12, pp. 6834–6848, Dec. 2023.

[39] A. Hareedy, S. Zheng, P. Siegel, and R. Calderbank, “Read-and-run constrained coding for modern Flash devices,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Seoul, South Korea, May 2022, pp. 1–6.

[40] A. Hareedy, S. Zheng, P. Siegel, and R. Calderbank, “Read-and-run constrained coding for modern Flash memories,” in *Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024.

[41] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, July 2016, pp. 770–778.

[42] F. Hemmati and D. J. Costello, “Truncation error probability in Viterbi decoding,” *IEEE Trans. Commun.*, vol. 25, no. 5, pp. 530–532, May. 1977.

[43] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” in *Proc. Neural Inf. Process. Syst. (NIPS)*, Dec. 2020.

[44] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, Dec, 1997.

[45] K. Huang, P. H. Siegel, and A. Jiang, “Functional error correction for robust neural networks,” *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 267–276, May 2020.

[46] P. Huang, Y. Liu, X. Zhang, P. H. Siegel, E. F. Haratsch, “Syndrome-coupled rate-compatible error-correcting codes: theory and application,” *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2311–2330, Jan. 2020.

[47] K. A. S. Immink, “Modulation systems for digital audio discs with optical readout,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Atlanta, Georgia, USA, Mar.–Apr. 1981, pp. 587–589.

[48] K. A. S. Immink and K. Cai, “Properties and constructions of constrained codes for DNA-based data storage,” *IEEE Access*, vol. 8, pp. 49523–49531, 2020.

[49] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, “Codes for digital recorders,” *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[50] P. Isola, J-Y. Zhu, T. Zhou, A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, July 2017, pp. 1125–1134.

[51] A. Jiang and E. F. Haratsch, "Machine learning: Enabling and enabled by advances in storage and memory systems," *IEEE BITS the Inf. Theory Magazine*, Sep. 2023, pp. 1–12.

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

[53] S. Kang, Y. Gao, J. Jeong, S-J. Park, J-W Kim, J-S No, H. Jeon, J. Lee, S. Kim, H. Park, and A. No, "Generative adversarial networks for DNA storage channel simulator," *IEEE Access*, vol. 11, pp. 3781–3793.

[54] R. Karabed and N. Nazari, "Analysis of error sequences for PRML and EPRML signaling performed over Lorentzian channel," in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, London, U.K., Nov. 1996, pp. 368–373.

[55] R. Karabed and P. H. Siegel, "Matched spectral-null codes for partial-response channels," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 818–855, May. 1991.

[56] R. Karabed, P. H. Siegel, and E. Soljanin, "Constrained coding for binary channels with high intersymbol interference," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1777–1797, Sep. 1999.

[57] N. Kashyap, R. M. Roth and P. H. Siegel, "The capacity of count-constrained ICI-free systems," in *IEEE Int. Symp. Inf. Theory*, Paris, France, Jul. 2019, pp. 1592–1596.

[58] A. Kato and K. Zeger, "On the capacity of two-dimensional run-length constrained channels," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1527–1540, Jul. 1999.

[59] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, Dec. 2018, pp. 9436–9446.

[60] Y. Kim and B. V. K. Vijaya Kumar, "Writing on dirty flash memory," in *Proc. 52nd Annu. Allerton Conf. Commun., Control, Comput.*, Monticello, IL, USA, Oct. 2014, pp. 513–520.

[61] D. P. Kingma and J. L. Bai, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representation (ICLR)*, San Diego, CA, USA, Dec. 2015.

[62] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Represent. Learn. (ICLR)*, Banff, Canada, Apr. 2014.

[63] R. Laroia, N. Farvardin, and S. A. Tretter, "On optimal shaping of multidimensional constellations," *IEEE Trans. Inf. Theory*, vol. 40, no. 4, pp. 1044–1056, Jul. 1994.

[64] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *Int. Conf. Mach. Learn. (ICML)*, New York, USA, June 2016.

[65] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Lett.*, vol. 23, no. 5, pp. 264–266, May 2002.

[66] A. Lenz, Y. Liu, C. Rashtchian, P. H. Siegel, A. Wachter-Zeh and E. Yaakobi, "Coding for efficient DNA synthesis," in *IEEE Int. Symp. Inf. Theory*, Los Angeles, CA, USA, Jul. 2020, pp. 2885–2890.

[67] Q. Li, A. Jiang, and E. F. Haratsch, "Noise modeling and capacity analysis for NAND flash memories," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Honolulu, HI, USA, June. 2014, pp. 2262–2266.

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

[69] S.-H. Lim, J.-B. Lee, G.-M. Kim and W. H. Ahn, "A stepwise rate-compatible LDPC and parity management in NAND flash memory-based storage devices," *IEEE Access*, vol. 8, pp. 162491–162506, 2020.

[70] Y. Liu, P. Huang, A. W. Bergman, and P. H. Siegel, "Rate-constrained shaping codes for structured sources," *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 5261–5281, Aug. 2020.

[71] Y. Liu, Y. Li, P. Huang, and P. H. Siegel, "Rate-constrained shaping codes for finite-state channels with cost," in *Proc. IEEE Int. Symp. Inf. Theory*, Espoo, Finland, June 2022, pp. 1354–1359.

[72] Y. Liu and P. H. Siegel, "Shaping codes for structured data," in *Proc. IEEE Global Commun. Conf.*, Washington, DC, USA, Dec. 2016.

[73] W. Liu, F. Wu, S. Meng, X. Chen, and C. Xie, "Error generation for 3D NAND flash memory," in *Proc. Design, Automation & Test in Europe Conf. & Exhib. (DATE)*, Antwerp, Belgium, Mar. 2022.

[74] W. Liu, F. Wu, M. Zhang, Y. Wang, Z. Lu, X. Lu, C. Xie, "Characterizing the reliability and threshold voltage shifting of 3D charge trap NAND flash," in *Proc. Design, Automation & Test in Europe Conf. & Exhib. (DATE)*, Florence, Italy, Mar. 2019.

[75] W. Liu, F. Wu, J. Zhou, M. Zhang, C. Yang, Z. Lu, Y. Yang, and C. Xie, "Modeling of threshold voltage distribution in 3D NAND flash memory," in *Proc. Design, Automation & Test in Europe Conf. & Exhib. (DATE)*, Grenoble, France, Feb. 2021.

[76] Y. Liu, S. Wu, and P. H. Siegel, "Bad Page Detector for NAND Flash Memory," in *Annual Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2020.

[77] Z. Liu, Y. Liu, and P. H. Siegel, "Generative modeling of NAND flash memory voltage level," in *Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2021.

[78] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch and O. Mutlu, "Enabling accurate and practical online flash channel modeling for modern MLC NAND flash memory," *IEEE J. Select. Areas Commun.*, vol. 34, no. 9, pp. 2294–2311, Sept. 2016.

[79] H. F. Löchel, M. Welzel, G. Hattab, A.-C. Hauschild, and D. Heider, "Fractal construction of constrained code words for DNA storage systems," *Nucleic Acids Research*, Mar. 2022, vol. 50, no. 5, page e30.

[80] B. H. Marcus, R. M. Roth, and P. H. Siegel. (Oct. 2001). *An Introduction to Coding for Constrained Systems.* [Online]. Available: https://personal.math.ubc.ca/~marcus/Handbook/

[81] B. H. Marcus, P. H. Siegel and J. K. Wolf, "Finite-state modulation codes for data storage," in *IEEE J. Sel. Areas Commun.*, vol. 10, no. 1, pp. 5–37, Jan. 1992.

[82] R. J. McEliece and I. M. Onyszchuk, "Truncation effects in Viterbi decoding," in *Proc. IEEE Military Communications Conference (MILCOM)'89*, Boston, MA, USA, Oct. 1989.

[83] B. E. Moision, "A truncation depth rule of thumb for convolutional codes," in *Proc. Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, Jan. 2008, pp. 555–557.

[84] B. E. Moision, "Constrained coding and detection for magnetic recording channels," Ph.D. Thesis, University of California, San Diego, La Jolla, CA, 2000.

[85] B. E. Moision and P. H. Siegel, "Distance enhancing constraints for noise predictive maximum likelihood detectors," in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, Sydney, Australia, Nov. 1998, pp. 2730-2735.

[86] B. E. Moision, P. H. Siegel, and E. Soljauin, "Distance-enhancing codes for digital recording," *IEEE Trans. Inf. Theory*, vol. 34, no. 1, pp. 69-74, Jan, 1998.

[87] J. Moon and W. Zeng, "Equalizer for maximum likelihood detectors," *IEEE Trans. Magn.*, vol.31, no.2, pp.1083-1088, Mar. 1995.

[88] R. Motwani, "Hierarchical constrained coding for floating-gate to floating-gate coupling mitigation in Flash memory," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Houston, TX, USA, Dec. 2011, pp. 1–5.

[89] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput.*, Monticello, IL, USA, Sep. 2016, pp. 341–346.

[90] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[91] T. Nguyen, K. Cai, K. A. Schouhamer Immink, and H. Mao Kiah, "Capacity-approaching constrained codes With error correction for DNA-based data storage", *IEEE Trans. Inf. Theory*, vol. 67, no. 8, pp. 5602–5613, Aug. 2021.

[92] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Columbus, OH, USA, June, 2014.

[93] T. Orekondy, A. Behboodi, and J. B. Soriaga, "MIMO-GAN: Generative MIMO channel modeling," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Seoul, South Korea, May 2022.

[94] N. Papandreou, H. Pozadis, T. Parnell, N. Loannoum, R. Pletka, S. Tomic, P. Breen, G. Tressler, A. Fry, and T. Fisher, "Characterization and analysis of bit errors in 3D TLC NAND flash memory," in *IEEE Int. Relilability Phys. Symp. (IRPS)*, Monterey, CA, USA, Apr. 2019.

[95] T. Parnell, N. Papandreou, T. Mittelholzer, and H. Pozidis,"Modelling of the threshold voltage distributions of sub-20nm NAND flash memory," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 2351–2356.

[96] B. Peleato, R. Agarwal, J. M. Cioffi, M. Qin, and P. H. Siegel, "Adaptive read thresholds for NAND flash," *IEEE Trans. Commun.*, vol. 63, no. 9, pp. 3069–3081, July. 2015.

[97] M. Qin, E. Yaakobi, and P. H. Siegel, "Constrained codes that mitigate inter-cell interference in read/write cycles for flash memories," *IEEE J. Select. Areas Commun.*, vol.32, no.5, pp. 836–846, May 2014.

[98] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshop*, Columbus, OH, USA, June 2014.

[99] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: convolutional networks for biomedical image segmentation," in *Int. Conf. Med. Imag. Comput. Comput.-assisted Intervention (MICCAI 2015)*, Springer, Cham.

[100] J. Saadé, A. Goulahsen, A. Picco, J. Huloux, and F. Pétrot, "Low overhead, DC-balanced and run length limited line coding," in *Proc. IEEE 19th Workshop on Signal and Power Integrity (SPI)*, Berlin, Germany, May 2015, pp. 1–4.

[101] V. G. Satorras and M. Welling, "Neural enhanced belief propagation on factor graphs," in *Int. Conf. Artificial Intell. Stat.*, pp. 685–692, 2021.

[102] M. Schuster and K.K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[103] C. E. Shannon, "A mathematical theory of communication," Bell Sys. Tech. J., vol. 27, Oct. 1948.

[104] E. Sharon *et al.*, "Data shaping for improving endurance and reliability in sub-20 nm NAND," presented at the Flash Memory Summit, Santa Clara, CA, USA, Aug. 2014.

[105] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "Data-driven factor graphs for deep symbol detection," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Los Angeles, CA, USA, June, 2020.

[106] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "ViterbiNet: Symbol detection using a deep learning based Viterbi algorithm," in *IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Cannes, France, July 2019, pp. 3319-3331.

[107] P. H. Siegel, "Constrained Codes for Multilevel Flash Memory," presented at *North American School of Information Theory (Padovani Lecture)*, La Jolla, California, Aug. 12, 2015.
Available:
http://cmrr-star.ucsd.edu/static/presentations/Padovani_Lecture_NASIT_Website.pdf.
Video: https://www.youtube.com/watch?v=FCv2PJryUr4.

[108] K. Sohn, H. Lee, and X. Yan "Learning structured output representation using deep conditional generative models," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Montréal, Canada, Dec. 2015.

[109] N. Sree Prem, "An Application of Machine Learning to Bad Page Prediction in Multilevel Flash," Master's Thesis, University of California San Diego, 2019.

[110] D. Tandler, S. Dörner, S. Cammerer, and S. T. Brink, "On recurrent neural networks for sequence-based processing in communications," in *2019 53rd Asilomar Conf. Signals, Syst., and Comput.*, Pacific Grove, CA, USA, Nov. 2019.

[111] D. T. Tang and R. L. Bahl, "Block codes for a class of constrained noiseless channels," *Inf. and Control*, vol. 17, no. 5, pp. 436–461, 1970.

[112] V. Taranalli, H. Uchikawa, and P. H. Siegel, "Error analysis and inter-cell interference mitigation in multi-level cell flash memories," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, UK, Jun. 2015, pp. 271–276.

[113] H. K. Thapar and A. M. Patel, "A class of partial response systems for increasing storage density in magnetic recording," *IEEE Trans. Magn.*, vol. 23, no. 5, pp.3666-3668, Sep. 1987.

[114] B. Varn, "Optimal variable length codes (arbitrary symbol cost and equal code word probability)," *Inf. Control*, vol. 19, no. 4, pp. 289–301, Nov. 1971.

[115] B. Vasic and E. Kurtas, *Coding and Signal Processing for Magnetic Recording Systems.* CRC Press, 2005.

[116] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.

[117] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260-269, Apr. 1967.

[118] J. Wang, T. Courtade, H. Shankar, and R. D. Wesel, "Soft information for LDPC decoding in Flash: Mutual-information optimized quantization," in *Proc. IEEE Global Commun. Conf.*, Houston, TX, USA, Jan. 2012.

[119] A. D. Weathers and J. K. Wolf, "A new rate 2/3 sliding block code for the (1,7) runlength constraint with the minimal number of encoder states," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 908-913, May. 1991.

[120] Y. Wang, C. Wu, L. Herranz, J. van de Weijer, A. Gonzalez-Garcia, and B. Raducanu, "Transferring GANs: generating images from limited data," in *Proc. European Conf. Comput. Vis. (ECCV)*, Munich, Germany, Sep. 2018.

[121] R. Wood, M. Williams, A. Kavcic, and J. Miles, "The feasibility of magnetic recording at 10 terabits per square inch on conventional media," *IEEE Trans. Magn.*, vol. 45, no. 2, pp. 917–923, Feb. 2009.

[122] Z. Wu, "Channel modeling, signal processing and coding for perpendicular magnetic recording," Ph.D. Thesis, University of California, San Diego, La Jolla, CA, 2009.

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

[124] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferrable are features in deep neural networks?" in *Proc. Neural Inf. Process. Syst. (NIPS)*, Montréal, Canada, Dec. 2014.

[125] N. Zheng, J. Li, S. Dahandeh, and T. Zhang, "Self-directed equalization for magnetic recording channels with multi-sensor read head," *IEEE Trans. Magn.*, vol. 52, no. 1, Jan. 2016.

[126] S. Zheng, C. Ho, W. Peng, and P. H. Siegel, "Flash-Gen: Spatio-temporal generator for flash memory systems," submitted to *IEEE Trans. Commun.*

[127] S. Zheng, C. Ho, W. Peng, and P. H. Siegel, "Spatio-temporal modeling for flash memory channels using conditional generative nets," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, Antwerp, Belgium, Apr. 2023.

[128] S. Zheng, Y. Liu, and P. H. Siegel, "PR-NN: RNN-based detection for coded partial-response channels," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 1967–1982, Jul. 2021.

[129] S. Zheng, Y. Liu, and P. H. Siegel, "RNN-based detection for coded partial-response channels," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Riva del Garda, Italy, Apr. 2021.

[130] S. Zheng and P. H. Siegel, “Code-aware storage channel modeling via machine learning,” in *Proc. IEEE Inf. Theory Workshop (ITW)*, Mumbai, India, Nov. 2022, pp. 196–201.

[131] S. Zheng, A. Tan, C. Fernández, I. G. Valenzuela, and P. H. Siegel, “Optimal shaping codes for a TLC flash memory,” in *Non-Volatile Memories Workshop (NVMW)*, La Jolla, CA, USA, Mar. 2024.

[132] J-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman, “Toward multimodal image-to-image translation,” in *Proc. Neural Inf. Process. Syst. (NIPS)*, Montréal, Canada, Dec. 2017, pp. 465–476.