

MACHINE LEARNING BASED PASSIVE TAP FOR AUTOMOTIVE ETHERNET

by

Maximus Ryan Marceau

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2026

©COPYRIGHT

by

Maximus Ryan Marceau

2026

All Rights Reserved

DEDICATION

I dedicate this thesis to all my family and friends who have supported me along this journey.

ACKNOWLEDGEMENTS

The author acknowledges Dr. Bradley M. Whitaker, Dr. Ross K. Snider, Dr. Brock J. LaMeres. The author also acknowledges Keith Filler, Skyler Reid, Keith D. Mecham, Noah Netz, Stephen Ude, and Audrey Ochs for their collaboration on this research. In addition, the author acknowledges the faculty and staff of the Departments of Electrical and Computer Engineering, Physics, and Science Education at Montana State University.

This research is part of an ongoing collaboration between Idaho National Laboratory (INL) and Montana State University (MSU) to explore applications of machine learning to enhance protection of process control systems from cybersecurity threats. Funding for this collaboration is provided through the INL Laboratory Directed Research & Development (LDRD) Program under DOE Idaho Operations Office Contract DE-AC07-05ID14517.

TABLE OF CONTENTS

1. INTRODUCTION	1
Project Motivation	1
Research Questions	3
Contributions	3
Thesis Organization	4
2. BACKGROUND	5
Introduction	5
Modern Day Communication	5
Receiver Pipeline	8
Communication Protocols	11
MODBUS	12
100BASE-TX	13
100BASE-T1	13
1000BASE-T	14
Machine Learning	15
Overview	15
Deep Learning	15
RNNs	16
LSTMs	17
GRUs	19
Bidirectional Long Short Term Memory (Bi-LSTM)	20
Current Machine Learning Applications in Communication	21
3. EXPERIMENT FRAMEWORK	23
Data Collection	23
PicoScope Decode Results	25
4. DECODING	27
Introduction	27
Preprocessing and Identifying Potential Frames	28
Clock Alignment	31
Identifying Clock Alignment Markers	31
Batched Clock Alignment	32
PAM3 Symbol Extraction	35
Frame Extraction	39
2T to 3B Conversion	41

TABLE OF CONTENTS – CONTINUED

Descrambling.....	43
Link Negotiation	43
Scrambling Method	44
Target Sequence to Solve	46
Solving for LFSR Current State	47
Descrambling Remaining Scrambled Sequence	51
Reorganize Bytes.....	52
Summary	53
Cyclic Redundancy Check	53
Results.....	55
Testing Methodology	55
HPF Tuning.....	57
Final Results and Impact.....	58
5. PASSIVE DOUBLE TAP	61
Introduction	61
Preparing Data for Training	61
Model Training.....	62
Model Predictions.....	66
Evaluation Metric.....	66
Overlapping Predictions.....	66
Comparing to Ground Truth	69
Determining RNN Architecture	70
Tuning Hyper-Parameters	72
Bi-LSTM Layers.....	72
Hidden Units	74
Batch Size	74
Segment Size.....	75
Overlap %.....	76
Model Selection	77
Changing Tap Points Comparison.....	78
Decoding Results	79
6. PASSIVE SINGLE TAP	82
Introduction	82
Reference Model	82
Determining RNN Architecture	83
Tuning Hyper-Parameters	84

TABLE OF CONTENTS – CONTINUED

Architecture.....	84
Hidden Units	85
Batch Size	85
Segment Size.....	86
Prediction Overlap	87
Selected Model Configurations	88
Comparison Models	89
Decoding Results.....	90
7. CONCLUSION	93
Research Questions Revisited	94
Future Work.....	95
REFERENCES CITED.....	96

LIST OF TABLES

Table	Page
3.1 PicoScope Decode Results	26
3.2 PicoScope Decode Accuracy.....	26
4.1 2T/3B Conversion Table.....	41
4.2 HPF Sweep Accuracy Results	57
4.3 HPF Sweep Accuracy Results	59
5.1 Reference Model Hyper-Parameters.....	70
5.2 Model Type.....	71
5.3 Hyperparameter Sweep — Architecture	73
5.4 Hyperparameter Sweep — Hidden Units.....	74
5.5 Hyperparameter Sweep — Batch Size.....	75
5.6 Hyperparameter Sweep — Segment Size	76
5.7 Hyperparameter Sweep — Overlap.....	77
5.8 Models Selected Hyper-Parameters and Results	78
5.9 Double Tap Point and RNN Comparison Models.....	79
5.10 Passive Double Tap Decode Results	80
6.1 Single Tap Reference Model Hyper-Parameters	83
6.2 Single Tap Model Type Comparison.....	83
6.3 Single Tap — Modifying Hyper-Parameters: Architecture.....	84
6.4 Single Tap — Modifying Hyper-Parameters: Hidden Units	85
6.5 Single Tap — Modifying Hyper-Parameters: Batch Size	86
6.6 Single Tap — Modifying Hyper-Parameters: Segment Size.....	87
6.7 Single Tap — Optimizing Overlap.....	88
6.8 Single Tap — Selected Models Hyper-Parameters and Results	89
6.9 Single Tap Point and RNN Comparison Models	90

LIST OF TABLES – CONTINUED

Table	Page
6.10 Single Tap Decode Results.....	91

LIST OF FIGURES

Figure	Page
2.1 OSI Model with Data Link and Physical layers expanded to represent typical Ethernet applications	7
2.2 Half-Duplex and Full-Duplex Communication Between Two Endpoints	8
2.3 Typical Ethernet Receive Pipeline Block Diagram	9
2.4 Ethernet Frame Format and Associated Bytes per Section	12
2.5 RNN Architecture	17
2.6 LSTM Cell Architecture	19
2.7 GRU Cell Architecture	20
2.8 BiLSTM Layer Architecture.....	21
3.1 Experimental Setup for Ground Truth Acquisition	24
4.1 Decoding Pipeline Overview Flow Diagram.....	28
4.2 Process for Identifying Potential Frames Within the Signal.....	30
4.3 Upsampled Signal with Identified Clock Alignment Markers.....	32
4.4 Clock Alignment Before and After Optimization	33
4.5 Stitched Clock Showing Recovery Points.....	34
4.6 PAM3 Symbol Extraction Overview Flow Diagram	35
4.7 Determination of Voting Characteristics Flow Diagram	36
4.8 Symbol Placement Attempt 1 Flow Diagram	36
4.9 Symbol Placement Attempt 2 Flow Diagram	37
4.10 Symbol Placement Attempt 3 Flow Diagram	38
4.11 Symbol Placement Attempt 4 Flow Diagram	39
4.12 Frame Extraction Overview Flow Diagram	40
4.13 Scrambling Process for 3-Bit Sequence With Initial LFSR Filled	45
4.14 LFSR Feedback and Shift Example for Master Device	46
4.15 Determining Target Sequence to Solve LFSR	47

LIST OF FIGURES – CONTINUED

Figure	Page
4.16 Solving LFSR 1st Iteration	48
4.17 Solving LFSR 4th Iteration.....	49
4.18 Solving LFSR 7th Iteration.....	50
4.19 Solving LFSR 12th Iteration	50
4.20 Fully Solved LFSR After 17th Iteration	51
4.21 Descrambling Process	52
4.22 4B to 3B Formatting.....	53
5.1 Model Architecture	64
5.2 MAE Before Overlapping Predictions	67
5.3 MAE After Overlapping Predictions.....	68
5.4 MAE After Overlapping Predictions Per Sample	69
5.5 Reference Model Prediction Results.....	72

NOMENCLATURE

Adam: Adaptive Moment Estimation
ADC: Analog-to-Digital Converter
AI: Artificial Intelligence
ANN: Artificial Neural Networks
ASCII: American Standard Code for Information Interchange
Bi-LSTM: Bidirectional Long Short Term Memory
CNN-BiLSTM: Convolutional Neural Network - Bidirectional Long Short Term Memory
CRC: Cyclic Redundancy Check
DFE: Decision-Feedback Equalizers DSP: Digital Signal Processing
FCS: Frame Check Sequence
FFE: Feed-Forward Equalizers
FFT: Fast Fourier Transform
FIR: Finite-Impulse-Response
GRU: Gated Recurrent Units
HPF: High Pass Filter
HTTPS: Hypertext Transfer Protocol
INL: Idaho National Labs
LFSR: Linear Feedback Shift Register
LLC: Logical Link Control
LSB: Least Significant Bit
MSB: Most Significant Bit
LSTM: Long Short-Term Memory
MAC: Media Access Control
MDI: Medium Dependent Interface
MII: Media Independent Interface
ML: Machine Learning
MLT-3: Multi-Level Transmit-3
OSI: Open Systems Interconnection
PAM: Pulse Amplitude Modulation
PAM3: Pulse Amplitude Modulation with Three Levels
PAM5: Pulse Amplitude Modulation with Five Levels
PCS: Physical Coding Sublayer
PMA: Physical Medium Attachment
PLL: Phase-Locked Loop
RNN: Recurrent Neural Network
RTU: Remote Terminal Unit
SFD: Start-Frame-Delimiter
SSD: Start of Stream Delimiter
TCP: Transmission Control Protocol
UDP: User Datagram Protocol

ABSTRACT

Modern full-duplex communication systems such as Automotive Ethernet present a significant challenge for passive network monitoring. Because both endpoints transmit simultaneously over a shared physical medium, the signal observed at any point along the cable is a superposition of two component signals that cannot be directly separated without knowledge of what each endpoint is transmitting.

This thesis investigates the use of recurrent neural networks as a passive monitoring tool for full-duplex 100BASE-T1 Automotive Ethernet. A custom decoding pipeline compliant with the IEEE 802.3 standard was first developed and validated against ground-truth data, achieving a frame decode accuracy of 93.0%. This pipeline served as the primary evaluation framework for all machine learning experiments.

Two passive tap configurations were investigated. In the double-tap approach, BiLSTM, LSTM, and GRU models were trained to predict component signals from two passively observed full-duplex measurements. All well-configured models exceeded the ground-truth decode accuracy baseline, with LSTM and GRU models achieving the highest accuracies of 97.0% and 96.9% respectively, despite producing higher mean absolute error than the BiLSTM. This result demonstrates that waveform MAE alone is not a sufficient indicator of passive tap performance, and that frame decode accuracy is the more meaningful evaluation metric.

The single-tap investigation examined whether signal separation remains feasible from a single observation point. While more sensitive to model architecture and hyperparameter selection, the best single-tap configurations achieved mean absolute error comparable to double-tap results, suggesting that a single passive probe may be sufficient for reliable signal reconstruction with appropriate model design.

Overall, these results demonstrate that machine learning is a viable approach for passive monitoring of full-duplex Automotive Ethernet, enabling frame-level decoding from simple oscilloscope probes without participation in link negotiation or access to endpoint transmission data.

INTRODUCTION

Project Motivation

Modern communication systems rely heavily on Ethernet-based protocols to enable reliable, high-speed data exchange across a wide range of applications. Implementing standardized protocols allows new technologies to integrate easily with existing systems that operate under the same specifications. Although these protocols are standardized, the demand for higher communication speeds continues to grow. As a result, protocol designs have become increasingly complex, making them more difficult to monitor, diagnose, and troubleshoot.

Traditional Ethernet monitoring techniques typically rely on active network devices or passive hardware taps that must be inserted directly into the communication link. While these approaches can be effective in many environments, they are not always scalable or cost-effective. In some communication systems, passive diagnostic methods are unavailable, requiring the use of active taps that can interfere with signal integrity.

This challenge becomes particularly significant in modern full-duplex Ethernet systems such as Automotive Ethernet. In half-duplex communication systems, passive monitoring is relatively straightforward because only one endpoint transmits at a time, allowing a probe to directly observe and decode the transmitted data. In contrast, full-duplex communication allows both endpoints to transmit simultaneously over the same physical medium. As a result, the observed signal at any point along the cable is a superposition of the signals transmitted by both endpoints. Each endpoint can recover the desired signal by subtracting its own transmitted waveform through echo cancellation; however, a passive observer does not have access to this information, making separation of the component signals significantly

more challenging.

This limitation has motivated research into alternative approaches for extracting information from full-duplex communication channels. Recent work has explored the use of machine learning, particularly recurrent neural networks (RNNs), which have demonstrated the ability to learn temporal dependencies in time-series signals. These models have been successfully applied to a variety of communication tasks, including channel modeling, interference mitigation, and network traffic analysis[2].

Prior work related to this project investigated the use of Long Short-Term Memory (LSTM) networks to predict the individual component signals within a full-duplex Automotive Ethernet link[11]. Using supervised training and two tap points on the communication channel, the model was able to learn the temporal dependencies of the full-duplex signals and predict the component signals with promising accuracy. These results suggest that machine learning could enable a new form of passive network tap capable of separating and decoding signals without requiring knowledge of the transmitted signals at the endpoints or the insertion of complex hardware. In principle, this approach could allow simple oscilloscope probes to monitor the communication channel while software performs decoding and diagnostic analysis. However, the prior work demonstrated only that the component signals could be approximated; it did not determine whether the predicted signals contained sufficient information to be successfully decoded into valid Ethernet frames.

Building upon this foundation, the goal of this research is to investigate whether machine learning techniques can be used to recover and decode Ethernet frames from a passively observed full-duplex Automotive Ethernet signal. Specifically, this work evaluates the feasibility of using RNN architectures to predict the individual component signals and determine whether the resulting predictions contain sufficient information to reconstruct valid Ethernet frames.

Research Questions

To evaluate the capability of machine learning as a passive monitoring approach for full-duplex Automotive Ethernet communication, several key research questions must be addressed. Previous work demonstrated that LSTM networks can approximate the individual component signals within a full-duplex Automotive Ethernet channel; however, it remains unclear whether these predictions are sufficiently accurate to enable reliable frame decoding. Furthermore, the performance of different neural network architectures and their ability to predict component signals within a full-duplex link must be examined. Addressing these questions is essential to determine whether machine learning can serve as a practical alternative to traditional hardware-based monitoring techniques.

This research investigates the following questions:

- Can machine learning models predict component signals within a full-duplex Automotive Ethernet channel using only passively observed full-duplex signals?
- Which recurrent neural network (RNN) architecture provides the most effective performance for predicting component signals in a full-duplex Automotive Ethernet link?
- Are the predicted component signals sufficiently accurate to allow successful decoding of Automotive Ethernet frames?
- What limitations exist when applying machine learning to passive monitoring of full-duplex communication channels?

Contributions

To address the primary research questions, the following contributions are made in this thesis:

- Development of a decoding pipeline to evaluate the accuracy of predicted signals based on successful Ethernet frame reconstruction.
- Evaluation of multiple recurrent neural network (RNN) architectures for predicting component signals in a full-duplex Automotive Ethernet channel, including a comparison of double-tap and single-tap configurations to assess the minimum passive observation required for reliable frame decoding.

Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides background on Ethernet communication systems, receiver pipelines, and machine learning techniques relevant to this work. Chapter 3 describes the design and implementation of the decoding pipeline, including validation using ground-truth data to evaluate prediction accuracy. Chapter 4 presents the double-tap signal prediction study, detailing the neural network architectures considered, hyperparameter tuning, selected models, and evaluation results based on both signal replication and Ethernet frame decoding accuracy. Chapter 5 investigates the single-tap signal prediction approach using a similar methodology, with the goal of exploring the limitations of machine learning models when only a single passive observation point is available. Finally, Chapter 6 summarizes the conclusions of this work and discusses potential directions for future research.

BACKGROUND

Introduction

This section provides an overview of modern-day communication protocols related to Ethernet, the current state of machine learning, particularly recurrent neural networks, and current applications of machine learning in Ethernet communication protocols.

Modern Day Communication

The Open Systems Interconnection (OSI) model is a conceptual framework that standardizes the functions of telecommunication and computing systems [3, 22]. It is divided into seven layers, each responsible for specific aspects of data communication.

Looking at the higher layers of the OSI model, brief descriptions are provided to give a clearer picture of how the model is structured and to situate this project within the broader networking framework. From the application to the transport layer, these layers operate at a higher level of abstraction than what will be directly addressed in this project. The application layer defines how a device transmits and receives data using a specific protocol, such as the Hypertext Transfer Protocol (HTTPS) [6]. The presentation layer handles data encryption, decryption, and compression, preparing information for transmission [22]. Meanwhile, the session layer establishes, manages, and terminates connections between two devices, ensuring reliable data transport [22].

At the transport layer, data is transferred between devices using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP). TCP is connection-oriented: a session is established between endpoints, and acknowledgments confirm the successful delivery of data, making TCP reliable but slower [37]. In contrast, UDP is connectionless; data is transmitted without waiting for acknowledgments, making it faster but less reliable

and suitable for devices or applications where occasional data loss is acceptable [37].

The network layer routes data packets from a source to a destination using IP addressing, where each device has a unique identifier [22]. Beneath this, the data link layer formats data into frames and assigns a unique Media Access Control (MAC) address. It is responsible for flow control, synchronization of endpoints, and error detection [3]. While this layer provides essential context for understanding end-to-end communication within the OSI model, its direct involvement in the implementation and analysis of this project is limited to associated terminology.

In typical Ethernet implementations, the data link layer consists of two sublayers: the Logical Link Control (LLC) and the MAC layer. The LLC ensures that a reliable link is established between endpoints using techniques such as echo cancellation, clock recovery, and, in some cases, scrambler lock. The MAC layer formats the data into frames, while the Reconciliation layer prepares these frames for transfer through the Media Independent Interface (MII) to the physical layer [1].

The PHY, part of the physical layer, consists of two main components: the Physical Coding Sublayer (PCS) and the Physical Medium Attachment (PMA). The PCS is responsible for encoding and decoding data, converting hexadecimal data into a binary bitstream. The PMA then converts this bitstream into the physical signal transmitted through the Medium Dependent Interface (MDI) and across the physical medium, such as copper cables or fiber optics [1].

This project primarily focuses on the data link and PHY layers. How the physical signal is received and the digital signal processing that takes place to extract information from that signal are at the core of what this project analyzes.

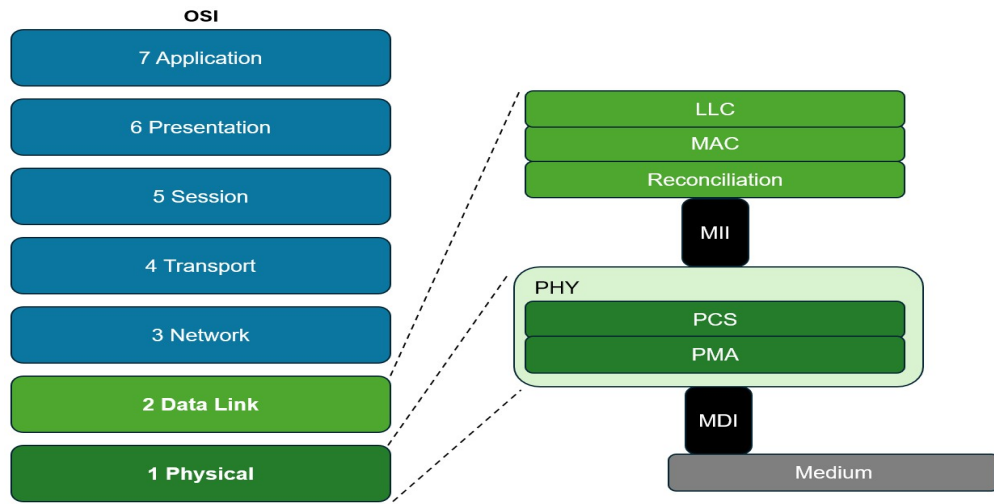


Figure 2.1: OSI Model with Data Link and Physical layers expanded to represent typical Ethernet applications

The distinction between half-duplex and full-duplex communication is a fundamental characteristic of data transmission protocols. In a half-duplex system, both endpoints share a common transmission medium and possess the capability to transmit and receive data; however, transmission occurs in only one direction at a time. This necessitates a coordination mechanism to prevent simultaneous transmissions. In contrast, a full-duplex system enables both endpoints to transmit and receive concurrently over the same physical medium, effectively doubling the potential communication throughput [45].

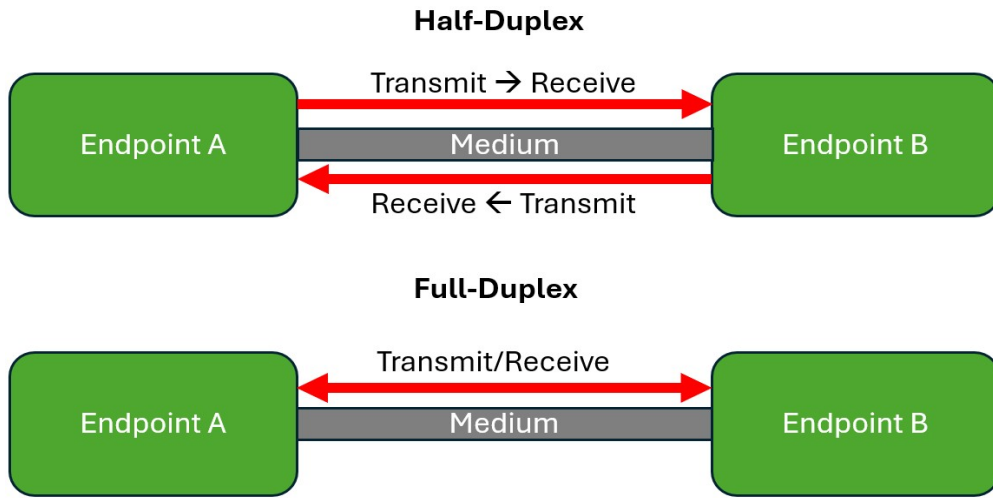


Figure 2.2: Half-Duplex and Full-Duplex Communication Between Two Endpoints

When observing the signal at an intermediate point along a half-duplex channel, only one signal is detected at any given moment, corresponding to the endpoint currently transmitting. Conversely, in a full-duplex system, the observed signal represents a superimposed combination of the signals transmitted by both endpoints. Despite this overlap, each endpoint can accurately recover the data transmitted by the other because it possesses knowledge of its own outgoing signal. By mathematically subtracting its own transmitted component from the composite received signal, an endpoint isolates and reconstructs the information sent from the opposite endpoint. This process is often referred to as echo cancellation [26].

Receiver Pipeline

Receivers in a network environment employ many different strategies to detect and decode the messages sent down the cable. In general, there is a typical flow for how this process works. There is typically analog front-end circuitry that obtains the signal and conditions it. This signal is converted into a digital signal via some form of analog-to-digital

converter (ADC). Echo cancellation is then deployed in full-duplex networks to extract the received signal, and adaptive equalization tunes the digital signal to mitigate transmission effects. Timing recovery examines the conditioned digital signal and determines the optimal samples that should be used to identify the received symbol. The decoding algorithm uses those optimal samples to extract a symbol stream. Within that symbol stream, encoded Ethernet frames must be decoded using an algorithm specific to the communication protocol in use.

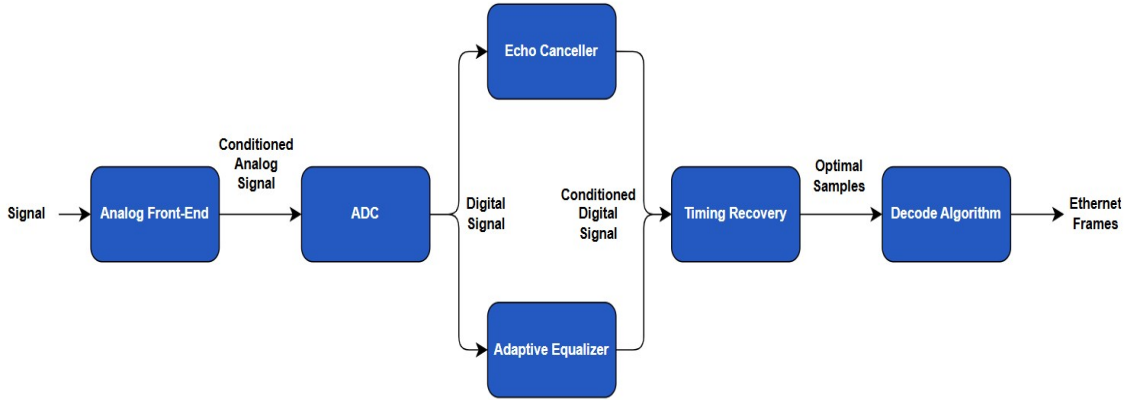


Figure 2.3: Typical Ethernet Receive Pipeline Block Diagram

Starting with the analog front-end, there is typically a differential voltage present on the communication medium. This stage typically includes impedance-matched termination, protection circuitry, and differential amplification to reject common-mode noise while preserving the received signal. A programmable gain amplifier or automatic gain control loop is commonly employed to scale the signal amplitude to the optimal range for digitization via an ADC. Continuous-time filtering may also be applied to limit out-of-band noise and prevent aliasing during sampling. Collectively, these circuits ensure that the received waveform is properly scaled, bandwidth-limited, and noise-conditioned before digital signal processing (DSP) stages are applied [17, 36].

Following analog front-end conditioning, the received signal is digitized using an ADC. The ADC samples the filtered analog waveform at a rate sufficient to capture enough samples per symbol for timing recovery to be achieved. Accurate sampling is critical because the following stages operate entirely in the digital domain [49]. Oversampling is frequently employed in communication receivers to improve robustness to timing offsets, enable digital filtering, and enhance signal reconstruction performance [27].

In full-duplex wired communication systems, the receiver must often operate while the transmitter is simultaneously active on the same communication medium. As a result, the signal couples into a superimposed signal containing the transmissions from both endpoints. Echo cancellation is employed to mitigate this interference by generating a digital estimate of the locally transmitted signal, filtering it according to the estimated channel response, and subtracting this prediction from the received signal. The cancellation filter is typically adaptive, allowing it to track variations in the coupling path caused by various factors. By removing the self-interference component of the received signal, echo cancellation enables reliable recovery of the desired remote signal prior to further processing [16, 29].

Running in parallel with echo cancellation, adaptive equalization is typically employed to mitigate effects on the signal such as channel loss, impedance discontinuities, and frequency-dependent attenuation introduced by the cable and connectors. The adaptive equalizer employs some form of filter, such as a finite-impulse-response (FIR) filter. In high-speed Ethernet PHYs, both feed-forward equalizers (FFE) and decision-feedback equalizers (DFE) are commonly used [15, 28].

Once the digital signal has undergone echo cancellation to provide the received signal, and adaptive equalization has ensured signal integrity, timing recovery can be implemented. The receiver must determine the optimal sampling instants for symbol detection. Because Ethernet signals are transmitted without a separate clock line, the receiver performs timing recovery to estimate the symbol period and phase directly from the received waveform. This

is typically achieved using a timing error detector that measures deviations between expected and observed signal transitions, combined with a feedback loop that adjusts the sampling clock or interpolation phase [30]. Common timing recovery approaches for modulated waveforms in digital synchronous communication are either inductive or deductive and typically involve a phase-locked loop (PLL) [24, 55]. The recovered timing ensures that samples are taken near the center of each symbol interval, where inter-symbol interference and noise sensitivity are minimized, thereby improving symbol extraction. This is often observed when plotting an eye diagram, where the eye opens up, or in a constellation diagram, where the clusters become smaller [56].

Once symbols have been extracted from the digitized and conditioned signal, the message can be decoded. The signal uses some form of modulation, which for full-duplex Ethernet is typically Pulse Amplitude Modulation (PAM) [33]. The receiver evaluates the samples identified by timing recovery and determines the symbol to extract. The receiver performs this process continuously, forming a symbol stream that contains encoded Ethernet frames [1]. Encoding schemes vary depending on the protocol. A decoding algorithm designed for the specific protocol processes the received symbols to determine the transmitted Ethernet frames [8].

Communication Protocols

Over the years, numerous communication protocols, both active and obsolete, have been developed to enable interoperability among devices from different manufacturers. These protocols provide standardized rules for data exchange, ensuring that networked components can communicate effectively when they adhere to the same specifications. Discussed below are four protocols—two half-duplex and two full-duplex—that are relevant to this research. This project specifically examined these protocols, starting with the simplest and progressing toward the more complex.

An important specification for Ethernet protocols is that the frames containing data transmitted between endpoints follow a format defined by the IEEE Standard [1]. Ethernet frames begin with a preamble, a set sequence of bytes identical for all Ethernet frames. The preamble consists of 7 bytes of "55" in hex, followed by the Start-Frame-Delimiter (SFD), a single byte "D5" in hex. Following these bytes are 6 bytes each for the MAC destination and source addresses, identifying where the frame is coming from and where it is going. The length or type field follows the addresses, consisting of two bytes that define either the Ethernet protocol or the frame length. Then, the data payload is transmitted, ranging from 46 to 1500 bytes. Finally, a cyclic redundancy check (CRC) validates whether any errors occurred during transmission or reception.

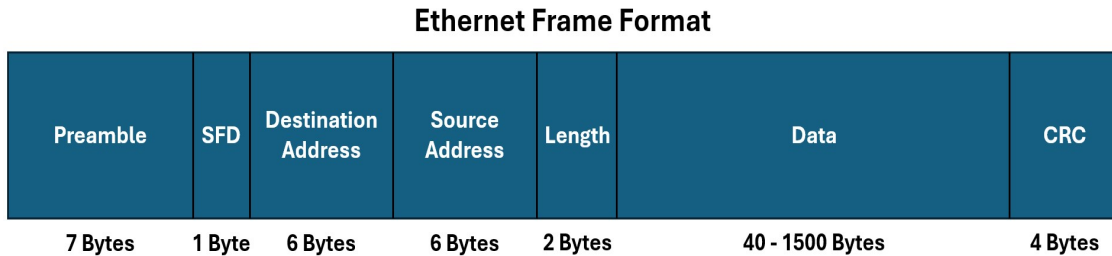


Figure 2.4: Ethernet Frame Format and Associated Bytes per Section

MODBUS

To establish a foundation for understanding these communication systems, this research began with the study of a simple serial protocol known as MODBUS. Developed by Modicon in 1979, MODBUS was one of the earliest and most widely adopted industrial automation protocols. It operates on a master/slave architecture, where the master device issues read or write queries, either to individual slaves or as a broadcast to all, while slave devices respond only to queries addressed to them [9].

MODBUS supports multiple data formats, including American Standard Code for Information Interchange (ASCII), Remote Terminal Unit (RTU), and TCP for Ethernet-

based communication. Focusing on the simpler serial RTU mode, data is transmitted as frames composed of a device address, function code, data field, and error-checking information. Each byte in a frame is sent as an 11-bit sequence: a start bit, eight data bits, one parity bit, and a stop bit. These bits are transmitted over a twisted-pair wire, with voltage levels representing binary 1s and 0s. Between transmissions, the line is held at a constant voltage to indicate an idle state. MODBUS is strictly half-duplex [47].

100BASE-TX

The next protocol examined was 100BASE-TX, commonly known as Fast Ethernet, a half-duplex communication standard operating at 100 Mbps. Data is transmitted over a single twisted-pair cable as voltage differentials and formatted into standard Ethernet frames. Before transmission, each frame undergoes several encoding steps. First, the nibbles (4-bit groups) within each byte are swapped. Each nibble is then converted into a 5-bit sequence using the 4B/5B lookup table, which also defines special control symbols such as start-of-frame and end-of-frame markers. The resulting bitstream is scrambled using a linear feedback shift register (LFSR), with XOR operations applied to specific taps defined by a common polynomial. This ensures data randomness and reduces electromagnetic interference. Finally, the scrambled data is converted to Multi-Level Transmit-3 (MLT-3) signaling, which uses three voltage levels (-1, 0, and +1). A logical 0 maintains the current level, while a logical 1 triggers a transition to the next level in a cyclical sequence ($0 \rightarrow 1 \rightarrow 0 \rightarrow -1 \rightarrow 0 \rightarrow 1 \dots$). This scheme allows efficient 100 Mbps transmission while maintaining low signal bandwidth. As with MODBUS, Fast Ethernet's half-duplex nature makes passive tapping relatively simple [1, 5, 10, 18, 21].

100BASE-T1

The primary focus of this research, however, is Automotive Ethernet (100BASE-T1). Automotive Ethernet was designed by Broadcom and referred to as BroadR-

Reach technology. Unlike traditional Ethernet variants, 100BASE-T1 supports full-duplex communication over a single twisted pair, enabling simultaneous bidirectional data transfer. This architectural difference makes the development of passive taps significantly more challenging. Directional couplers can serve as a form of passive tap, as they are minimally disruptive and allow the main link to remain active [44].

At a high level, Automotive Ethernet uses the standard Ethernet frame structure but employs a distinct encoding process. Each byte of data is divided into 3-bit segments, which are then scrambled using an LFSR whose initial state is negotiated during link establishment. The scrambling operation applies XOR feedback according to a defined polynomial [1]. The resulting scrambled 3-bit sequences are mapped to ternary symbols (-1, 0, +1) via a lookup table and transmitted as corresponding voltage levels over the twisted pair. This encoding method is known as PAM3 (Pulse Amplitude Modulation with Three Levels) [7, 13, 38].

1000BASE-T

Finally, Gigabit Ethernet, also known as 1000BASE-T, builds upon the principles of Automotive Ethernet but increases data throughput by transmitting over four twisted-pair cables simultaneously. It employs PAM5 (Pulse Amplitude Modulation with Five Levels) encoding, which uses five voltage levels (-2, -1, 0, +1, +2). Each pair transmits data at 100 Mbps; however, by sending four symbols in parallel—one on each pair—an entire byte can be conveyed per transmission cycle. This parallelism, combined with higher symbol resolution, enables Gigabit Ethernet's data rate of 1 Gbps while maintaining signal integrity across multiple wire pairs. Gigabit Ethernet also employs a side-stream additive scrambler to ensure randomness. It uses a similar LFSR whose initial state is negotiated during link establishment, with additional operations to scramble a full byte of data at a time [1, 4, 14]. Upon successful development of a machine learning (ML) based passive tap for Automotive Ethernet, this protocol would be the next target for investigation.

Machine Learning

Overview

ML is a field of Artificial Intelligence (AI) that enables computer systems to learn from data by identifying patterns and making decisions using statistical models, which are refined to improve accuracy and performance when predicting or performing certain tasks [14].

ML is divided into four categories: supervised, unsupervised, semi-supervised, and reinforcement learning. Supervised learning requires labeled data partitioned into training and test sets. Models are typically used for tasks such as classification or regression [34]. Unsupervised learning involves training a model on unlabeled data, with the goal of inferring patterns. These models are commonly used for clustering or dimensionality reduction [31]. Semi-supervised learning combines a small amount of labeled data with a larger amount of unlabeled data to improve performance, often applied in medical imaging, text classification, and speech recognition [41]. Reinforcement learning involves a guided trial-and-error approach, where an agent receives positive feedback for decisions that move it closer to a desired outcome. These models are often applied in robotics and automation [19].

Deep Learning

Deep learning is a subset of ML inspired by biological neural systems. These models, referred to as Artificial Neural Networks (ANNs), are typically built with multiple layers of neurons. Each layer has an associated activation function, usually non-linear, allowing the model to learn complex patterns. The number of layers, nodes, and types of activation functions are hyperparameters that can be tuned [51].

In fully connected architectures, each neuron in one layer connects to every neuron in the next. Each connection has a weight, and each neuron has a bias applied to its weighted

input. During training, weights and biases are adjusted to improve performance. Most learning occurs in hidden layers, between the input and output layers [50, 52].

Backpropagation is used to train the model. During a forward pass, input predictors are fed into the network to produce an output. This output is compared to the ground truth to determine accuracy. The difference is quantified using a loss function. During backpropagation, the gradient of the loss with respect to each weight and bias is calculated, showing how changes affect the loss. Optimization algorithms, such as gradient descent, update the weights and biases to reduce the loss. This process repeats until training stabilizes or meets a convergence criterion [23].

RNNs

RNNs are a type of deep learning network designed to leverage memory from previous states to process sequences. An RNN cell uses shared weights across time steps, allowing it to retain information over sequences. A hidden state is calculated for each input to capture sequential dependencies, which is then used to determine the output [43].

Vanilla RNNs contain a single hidden layer and are suitable for modeling short-term dependencies. However, they can suffer from the vanishing gradient problem, where gradients shrink exponentially over time, potentially preventing learning of long-term dependencies [32].

Figure 2.5 below shows a typical RNN layer structure. An input, along with the hidden state from the previous time step, is passed through the RNN cell. The result is a new hidden state, which is then fed into the RNN cell for the next input. The new hidden state also serves as the output in sequence-to-sequence RNNs.

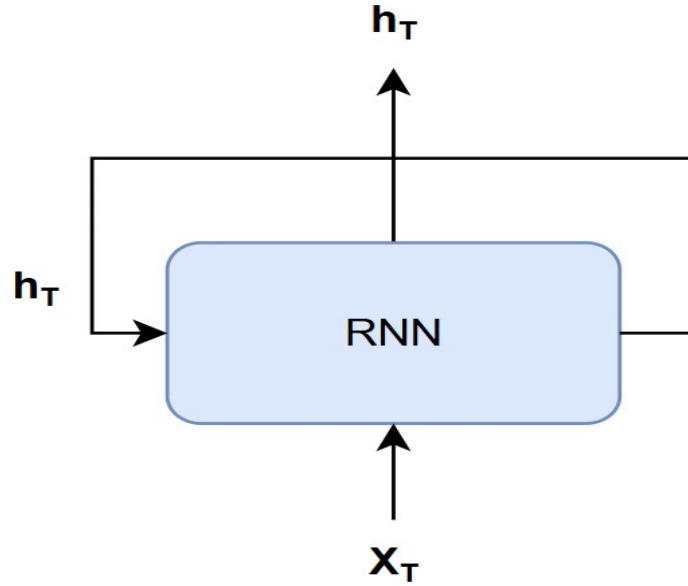


Figure 2.5: RNN Architecture

To address the vanishing gradient problem, LSTMs and Gated Recurrent Units (GRUs) were introduced. An LSTM introduces a memory mechanism that helps maintain long-term dependencies often lost in a vanilla RNN. A GRU has a similar goal but uses a simpler architecture, allowing for faster training while being slightly less effective at preserving long-term dependencies [54].

LSTMs

An LSTM network is organized into cells containing three primary gates: forget, input, and output. At each time step, the LSTM receives the current input from the sequence, the hidden state from the previous step, and the memory cell state [54].

The forget gate [48, 54] concatenates the hidden state and input vector and passes them through a sigmoid activation. The output determines the fraction of the previous memory that should be retained:

$$f_t = \sigma(W_f * [h_{t-1}, X_t] + b_f) \quad (2.1)$$

At the input gate [54], the concatenated hidden state and input are passed through both a sigmoid and a tanh activation function. The sigmoid output determines which portions of the memory should be updated, while the tanh output generates candidate values for the memory update. These candidate values are combined with the previous memory to update the cell state:

$$i_t = \sigma(W_i * [h_{t-1}, X_t] + b_i) \quad (2.2)$$

$$\hat{C}_t = \tanh(W_c * [h_{t-1}, X_t] + b_c) \quad (2.3)$$

The output gate [40, 54] determines the next hidden state. The input and hidden state are combined with learned weights and passed through a sigmoid activation. The updated cell state is passed through a tanh activation, and the outputs of the two paths are multiplied to form the new hidden state:

$$o_t = \sigma(W_o * [h_{t-1}, X_t] + b_o) \quad (2.4)$$

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \quad (2.5)$$

$$h_t = o_t * \tanh(C_t) \quad (2.6)$$

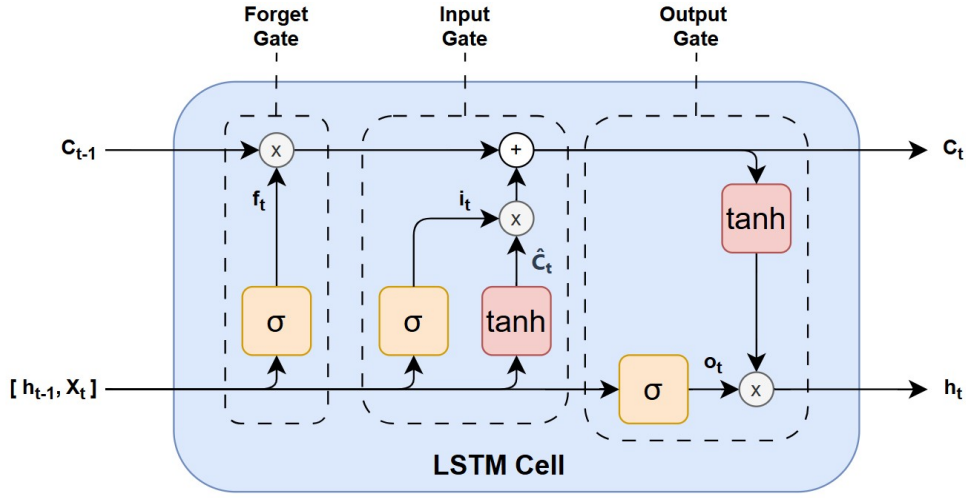


Figure 2.6: LSTM Cell Architecture

GRUs

The GRU cell is a simpler variant with two gates: reset and update. The reset gate determines how much of the previous hidden state contributes to the candidate state, while the update gate balances the contribution between the previous hidden state and the candidate state [54].

For the reset gate [39, 54], the previous hidden state is concatenated with the current input and passed through a sigmoid function. The output, r_t , is multiplied element-wise with the previous hidden state:

$$r_t = \sigma(W_r * [h_{t-1}, X_t] + b_r) \quad (2.7)$$

For the update gate [39, 54], the previous hidden state and input are concatenated and passed through a sigmoid to generate z_t . The previous hidden state is then combined with the candidate hidden state according to z_t to compute the new hidden state:

$$z_t = \sigma(W_z * [h_{t-1}, X_t] + b_z) \quad (2.8)$$

$$\hat{h}_t = \tanh(W_h * X_t + W_h * (r_t * h_{t-1} - 1) + b_h) \quad (2.9)$$

$$h_t = z_t * h_{t-1} + (1 - z_t) * \hat{h}_t \quad (2.10)$$

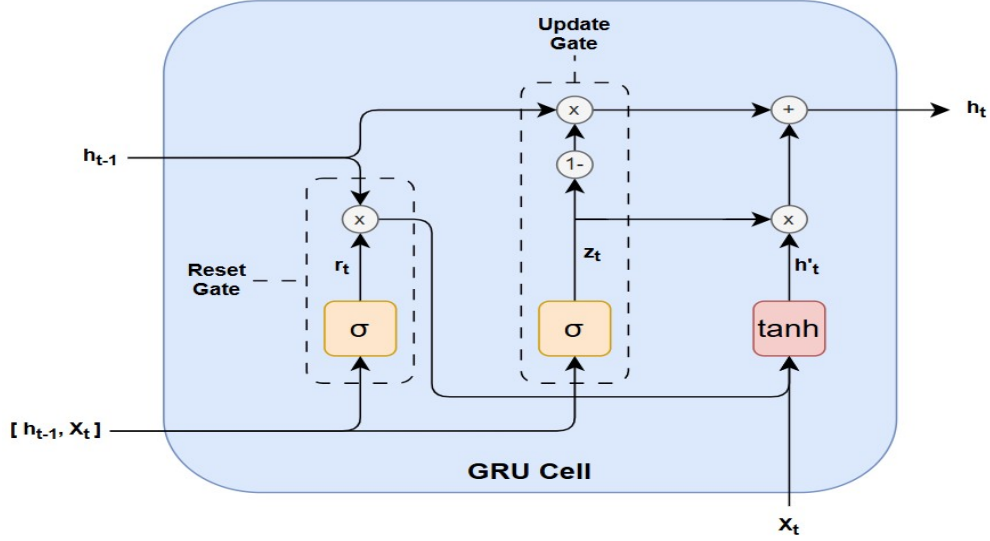


Figure 2.7: GRU Cell Architecture

Bidirectional Long Short Term Memory (Bi-LSTM)

The BiLSTM layer uses the same LSTM cell structure but consists of both a forward and a backward layer. This allows the network to learn dependencies from both past and future inputs at each time step. The input at the current time step is fed into the forward LSTM cell along with the initial cell state, which includes the short-term hidden state and long-term cell state. The cell is unrolled through the sequence from the first input to the last, producing the forward output. Simultaneously, the initial cell state is also fed into a backward LSTM, which processes the input sequence in reverse, from the last element to the first, producing the backward output. At each time step, the outputs from the forward and backward LSTMs are concatenated to form the final BiLSTM output. Therefore, each time step's output reflects both preceding and subsequent dependencies in the sequence [53, 54].

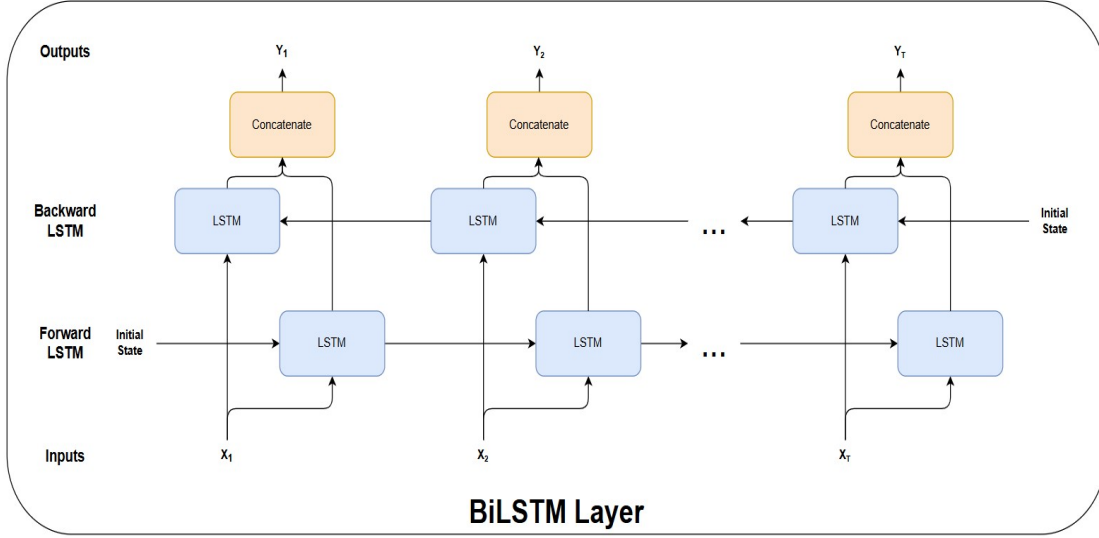


Figure 2.8: BiLSTM Layer Architecture

Current Machine Learning Applications in Communication

While traditional transceiver and communication protocol design rely on a combination of analog front-end circuitry and extensive DSP with control algorithms, ML is increasingly explored for applications in communication systems. ML has primarily been applied at the physical, link, and network layers, with use cases including improving DSP blocks in the PHY layer and recognizing patterns across the communication link [2, 42, 46]. These studies highlight ML's potential to complement or even replace conventional design elements in modern communication systems.

A foundational example involves framing a communication system as an end-to-end autoencoder, where the transmitter and receiver are jointly optimized through model training [35]. ANNs can replace or augment conventional blocks such as modulation, detection, and channel equalization. Deep learning models can learn efficient signal representations and detection strategies directly from data, showing that ML could replace specific DSP blocks in traditional communication design.

An example of ML applied to a specific DSP function at the PHY layer involved training an LSTM network to predict the frequency-hopping pattern in a satellite communication system [25]. The model learned the temporal dependencies of the hopping sequence and could anticipate future frequency changes, reducing reliance on conventional synchronization algorithms. This demonstrates that RNNs can effectively capture complex, time-dependent behaviors in communication signals.

ML has also been applied to Automotive Ethernet protocols. One study investigated how ML could enhance security by using a Convolutional Neural Network - Bidirectional Long Short Term Memory (CNN-BiLSTM) time series prediction model to learn the normal temporal behavior of network traffic [12]. Anomalies were detected by identifying deviations from expected patterns. This work demonstrates that ML techniques can monitor PHY-layer signals and recognize abnormal patterns at the network layer, showing utility beyond traditional DSP tasks.

ML can also support design optimization in communication networks. A study of a 10-Gbit/s passive optical access network explored how ML could determine optimal design parameters, such as fiber length and secure key rate, which are otherwise difficult to set efficiently [20]. The ML model predicted these parameters, guiding design decisions and improving network performance. This illustrates that ML can operate adjacent to the PHY layer to optimize system-level decisions, complementing conventional engineering approaches.

Finally, previous work on this project provides a direct foundation for this research. LSTM models were trained to predict the component signals of a full-duplex Automotive Ethernet link between two endpoints [11]. Prediction accuracy was sufficiently high that the resulting signals could potentially be decoded into frames, providing a proof-of-concept for an ML-based passive tap. This motivates the current project, which seeks to extend this approach and evaluate the feasibility of decoding predicted signals in a real-world full-duplex Ethernet system.

EXPERIMENT FRAMEWORK

Data Collection

Since the machine learning model is intended to use supervised learning, a ground truth must first be established within an operational network environment using Automotive Ethernet. Idaho National Labs (INL) conducted an experiment to gather data by tapping the cable connecting two communicating endpoints, referred to as Endpoint *A* and Endpoint *B*.

The objective of this experiment was to capture full-duplex signals and train a model capable of predicting the individual component signals transmitted by each endpoint. To achieve this, two endpoints were configured to communicate over a full-duplex Automotive Ethernet channel. Oscilloscope probes were placed at three locations along the channel: Probe *D* was positioned at the midpoint between the endpoints, Probe *A* was located between Endpoint *A* and Probe *D*, and Probe *B* was located between Endpoint *B* and Probe *D*.

All full-duplex signals were captured using a PicoScope oscilloscope. The PicoScope software is capable of using measurements from Probes *A* and *B*, along with known system parameters such as cable length and transmission speed, to extrapolate the individual component signals, denoted as T_A and T_B . While the exact internal method is proprietary, it effectively separates the superimposed full-duplex signal into its component signals.

The PicoScope sampled the full-duplex signals at 313 MHz, while the data transmission rate was 66 MHz. Automotive Ethernet utilizes PAM3, meaning each symbol corresponds to one of three voltage levels: -1 V, 0 V, or 1 V. Because the full-duplex signal is a superposition of two component signals, the observed signal exhibits voltage levels of approximately -2 V, 0 V, and 2 V.

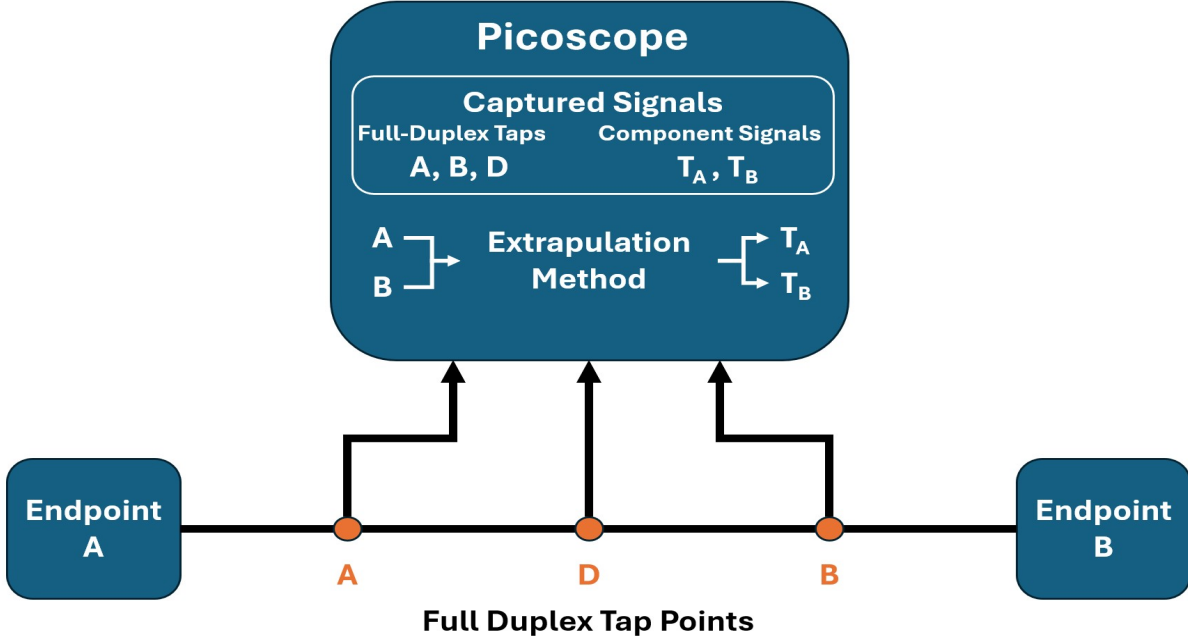


Figure 3.1: Experimental Setup for Ground Truth Acquisition

The experiment produced three full-duplex signals (A , B , and D), each sampled continuously at 313 MHz. The component signals T_A and T_B were extrapolated at the same effective sampling rate, corresponding to approximately 4.7 samples per symbol. However, unlike the full-duplex signals, the component signals were captured in discrete buffers rather than continuously. Each experiment consisted of 64 buffers of data. Within each buffer, full-duplex signals were continuously sampled, while the component signals were extrapolated for most of the buffer duration, except for small regions at the beginning and end where values were undefined (NaNs).

When combining buffers for analysis, these undefined regions introduced gaps in the component signal data. These gaps can impact decoding performance, as valid Ethernet frames may be partially or entirely contained within these missing segments.

Overall, the experiment produced five signals: three full-duplex signals measured at different points along the channel, and two extrapolated component signals corresponding to each endpoint transmission. The full-duplex signals represent continuous superimposed

PAM3 waveforms, while the component signals consist of PAM3 waveforms segmented into buffers with gaps between them.

PicoScope Decode Results

To assist in developing the decoding pipeline, the experiment was configured such that both endpoints repeatedly transmitted the same Ethernet frame. As a result, although many frames were present in the captured data, each component signal contained only one unique frame repeated multiple times.

The PicoScope software includes a built-in decoding algorithm that scans the component signals, identifies frame boundaries, and attempts to decode the transmitted data. The decoding results for ten experimental runs are summarized in Table 3.2 below. While a large number of frames were identified, only a small fraction were successfully decoded. The overall decoding accuracy was 8.6%, with T_A achieving 11.1% accuracy and T_B achieving 6.6%.

Table 3.1: PicoScope Decode Results

Experiment Capture	T_A VALID Frames	T_A Total Frames	T_B VALID Frames	T_B Total Frames
1	8	47	5	43
2	5	58	7	53
3	2	37	2	34
4	4	50	2	59
5	6	54	2	47
6	2	40	5	41
7	6	38	3	64
8	4	33	3	43
9	10	39	2	39
10	2	47	7	149
Total	49	443	38	572

Table 3.2: PicoScope Decode Accuracy

Overall Decode Accuracy	T_A Decode Accuracy	T_B Decode Accuracy
8.6%	11.1%	6.6%

These results indicate that while the PicoScope software can approximate the component signals, the resulting decode accuracy is relatively low. Therefore, the PicoScope alone does not provide a reliable passive decoding solution for this experiment. However, the extrapolated component signals may still serve as a valid ground truth for training and evaluation. To investigate this further, a custom decoding pipeline was developed to assess the true decodability of the predicted signals.

DECODING

Introduction

After the full-duplex signals have been separated into their respective component signals, each signal must be decoded. Successful decoding provides further validation that the signal separation process accurately recovered the original transmitted signals. Therefore, a decoder was designed to be compliant with the 100BASE-T1 standard for Automotive Ethernet [1], effectively functioning as a receiver within the communication system.

The encoding scheme used in this standard is defined in Clause 96 of the IEEE Ethernet specification. At the MAC layer, data is represented as a sequence of hexadecimal values, which are converted into binary form for transmission. This binary data is then encoded using a PAM3 modulation scheme, resulting in a signal with voltage levels ranging from -1 V to +1 V. The decoding process performs the inverse operation, taking a PAM3 signal as input and recovering the original hexadecimal data.

The following sections describe the decoding process in detail. At each stage, the corresponding encoding operation defined by the IEEE standard is first introduced, followed by the implementation of the decoding method. The decoding pipeline is presented in chronological order, while the encoding process is described in reverse order to provide proper context for each step.

Figure 4.1 illustrates the overall decoding pipeline. Beginning with the raw component signals, a preprocessing and search stage scans the data to identify segments where frames are likely present. For each candidate segment, clock alignment markers are detected and stored. A processing loop is then applied to each potential frame, where a clock recovery algorithm aligns the sampling clock to the signal, and a symbol extraction method recovers the PAM3 symbols.

Once a PAM3 symbol stream has been obtained, a frame detection step isolates the

portion of the stream corresponding to a valid frame. This segment is then converted into a scrambled binary sequence using the standard-defined 2T-to-3B mapping. The binary sequence is subsequently descrambled, reorganized, and validated using a CRC.

Throughout the decoding process, multiple validation checks are performed. If an error is detected at any stage, the algorithm returns to the clock recovery step to attempt improved alignment. This iterative process continues until the frame is successfully decoded or until no further improvement in clock recovery is possible, at which point the frame is classified as invalid.

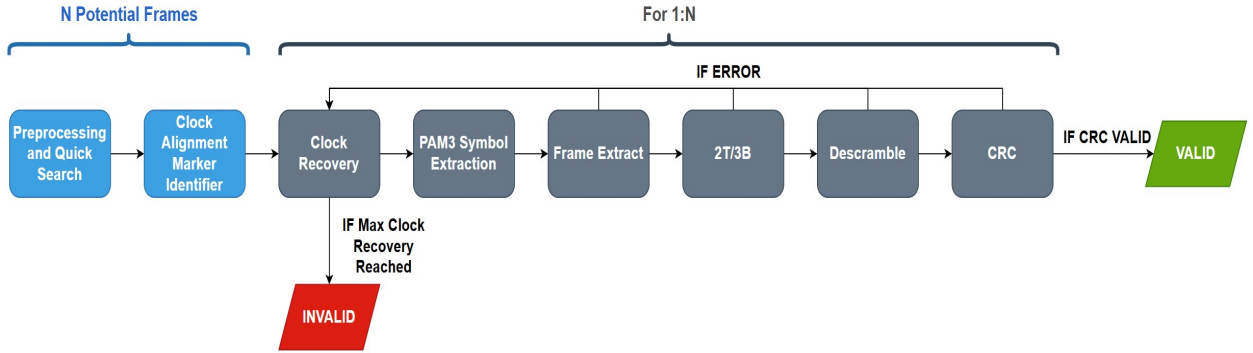


Figure 4.1: Decoding Pipeline Overview Flow Diagram

Preprocessing and Identifying Potential Frames

Most preprocessing steps were completed during the signal separation stage; however, additional preprocessing is required for decoding.

In Automotive Ethernet, frames are defined by unique sequences of PAM3 symbols that mark their beginning and end. These sequences are known as the Start of Stream Delimiter (SSD) and End of Stream Delimiter (ESD). The SSD and ESD occur only at the boundaries of a frame and should not appear elsewhere in the signal during idle transmission.

The SSD is defined as six consecutive zero symbols (000000) within the PAM3 signal. The ESD can take one of two forms: either four consecutive zero symbols followed by two consecutive "1" symbols, or four consecutive zero symbols followed by two consecutive "-1" symbols (000011 or 0000-1-1). A sequence of "1" symbols indicates that the transmitting endpoint considered the frame valid, while "-1" symbols indicate that the frame was marked invalid during transmission. These symbol representations are discussed further in Section 4.

To identify candidate frames and initiate the decoding process, a lightweight search function was developed. The purpose of this function is to efficiently scan the signal and extract segments likely to contain an SSD, thereby isolating potential frames for further processing.

This approach significantly reduces computational cost. As described in the following section, clock alignment is the most computationally intensive step in the decoding pipeline. By restricting clock alignment to only those regions where frames are likely present, the overall processing time is greatly reduced.

Because the signal is transmitted using PAM3 modulation, a simple threshold-based classification method is used to approximate symbol values. Each sample is classified as "1", "0", or "-1" based on its voltage level: samples above 0.3 V are classified as "1", samples between -0.3 V and 0.3 V are classified as "0", and samples below -0.3 V are classified as "-1".

The number of consecutive samples within each region is counted before a transition occurs. Each count is then divided by the estimated samples-per-symbol rate and rounded to the nearest integer, providing an approximate number of PAM3 symbols in each segment. This produces a coarse PAM3 representation of the signal. While this approximation is not suitable for accurate decoding, it is sufficient for identifying candidate SSD locations.

Because the SSD consists of six consecutive zero symbols and is unlikely to occur elsewhere in the signal, the search function identifies regions containing five or more

consecutive estimated zero symbols. Using five instead of six provides robustness against estimation error and reduces the likelihood of missing valid SSDs. The indices of these potential SSD locations are then recorded.

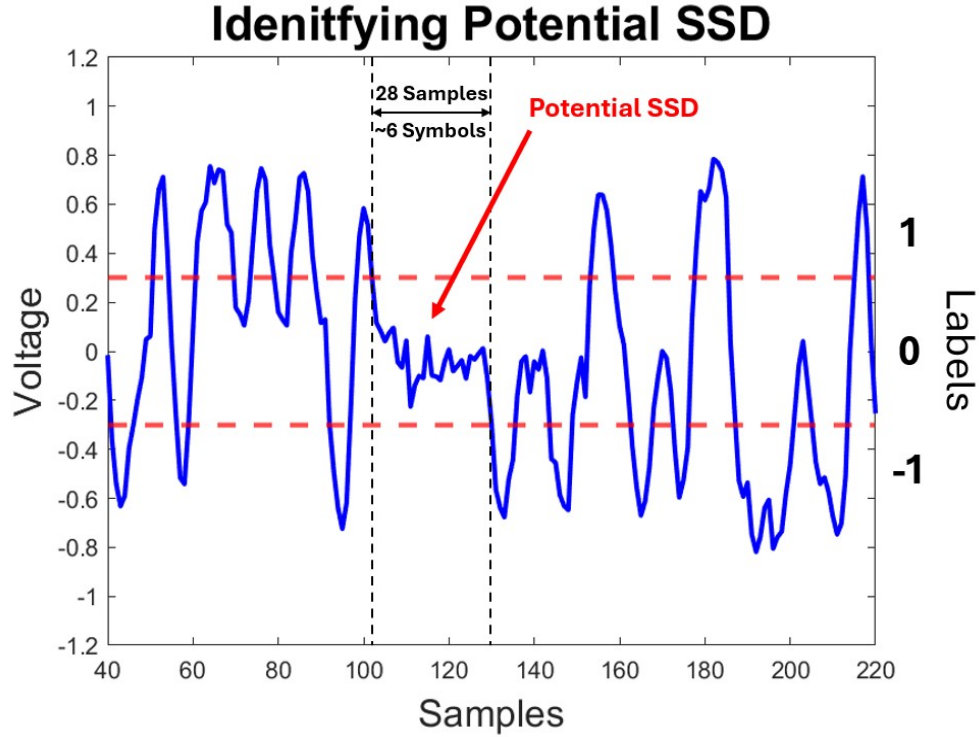


Figure 4.2: Process for Identifying Potential Frames Within the Signal

For each detected SSD location, a segment of the signal is extracted and stored as a separate array. The length of each segment is chosen to ensure that it contains enough samples to represent the longest possible Ethernet frame (1,518 bytes). Given that the encoding scheme maps 2 symbols to 3 bits and the signal contains approximately 4.7 samples per symbol, this results in approximately 38,376 samples for a maximum-length frame. Accordingly, for each potential SSD, 100 samples preceding the SSD and 38,376 samples following it are extracted.

Finally, each extracted segment is upsampled by a factor of 10. This improves the accuracy of subsequent processing steps, particularly clock alignment, where finer temporal

resolution enhances the detection of alignment markers. The result is a collection of upsampled signal segments that are likely to contain frames and are suitable for detailed decoding analysis.

Clock Alignment

Clock alignment is one of the most critical components of the decoding pipeline. Given a transmission rate of 66 MHz, a PAM3 symbol is transmitted every half cycle of a 33 MHz clock. Therefore, clock recovery involves aligning a 33 MHz square wave to the received signal in order to correctly sample each symbol.

Two primary functions were developed for clock recovery: (1) identification of clock alignment markers and (2) alignment of the clock to these markers.

Identifying Clock Alignment Markers

The first step is to identify reliable alignment markers within the signal. Ideally, these markers correspond to symbol transitions. Due to noise and signal variability, only the largest transitions, specifically transitions between -1 and 1, were found to be consistently reliable. Smaller transitions were more prone to ambiguity due to intermediate voltage levels and local extrema.

To detect these transitions, a coarse PAM3 classification is first performed using thresholding. Each sample is classified as -1, 0, or 1 using thresholds of -0.3 V and 0.3 V. Transitions between regions are then identified and categorized. Because the signal is upsampled, transitions between -1 and 1 always pass through the intermediate (0) region, allowing transition direction to be inferred.

Using these classified transitions, pattern matching is applied to identify candidate -1 to 1 and 1 to -1 transitions. Specifically, the patterns -1, 0, 1 and 1, 0, -1 are detected within the classified sequence. The index corresponding to the intermediate 0 value is extracted as

a candidate alignment marker.

However, not all such patterns correspond to true transitions. If the signal remains in the 0 region for an extended duration, the pattern may instead represent a valid 0 symbol rather than a transition. To address this, the duration of the 0 region is evaluated. If the number of consecutive samples in the 0 region exceeds a threshold, the candidate is rejected; otherwise, it is accepted as a valid transition marker. The resulting set of indices represents the alignment markers used for clock recovery.

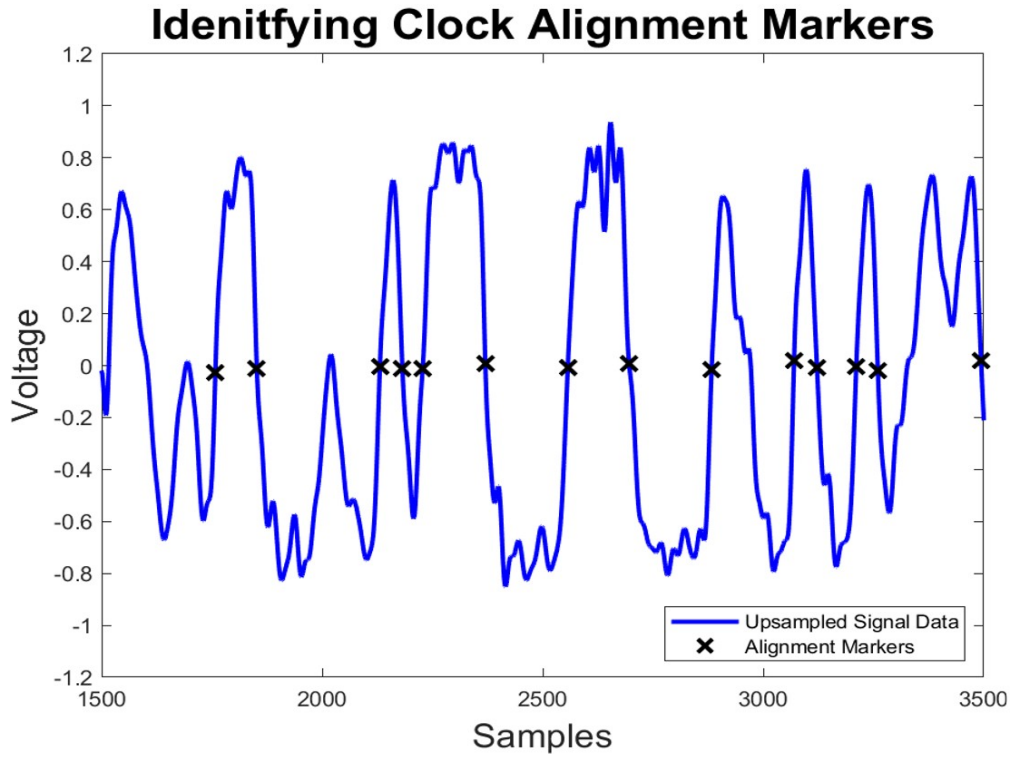


Figure 4.3: Upsampled Signal with Identified Clock Alignment Markers

Batched Clock Alignment

Once alignment markers are identified, the next step is to align the clock. Although the nominal symbol rate is known, small timing variations require continuous clock adjustment to maintain alignment with the signal.

To achieve this, the signal is divided into segments, each containing 10 alignment markers. This choice balances two competing factors: smaller segments allow more frequent clock recovery, while larger segments provide more markers for accurate alignment.

For each segment, a 33 MHz clock is generated. The goal is to align the clock transitions with the detected alignment markers. The alignment process iteratively shifts the phase of the clock and computes the average distance between the closest clock transition points and alignment markers. The phase that minimizes this average distance is selected as the optimal alignment for the segment.

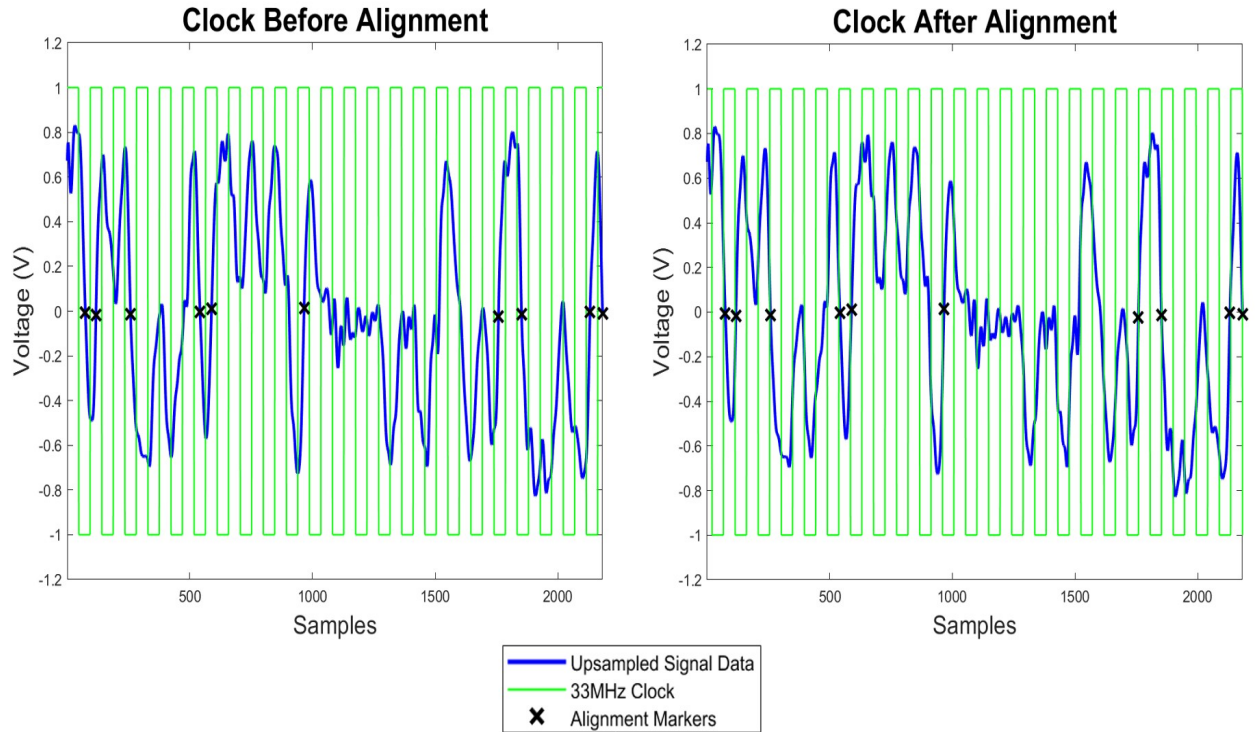


Figure 4.4: Clock Alignment Before and After Optimization

After determining the optimal clock for each segment, the segments are stitched together to form a continuous recovered clock. Due to variations in phase alignment, discontinuities can occur at segment boundaries, resulting in irregular high or low durations.

Short-duration segments are acceptable and indicate normal clock adjustment. However, excessively long segments can lead to symbol extraction errors. To mitigate this, the final sample of each segment is flipped to enforce a transition at the boundary. Additional checks ensure that this adjustment does not introduce extended high or low regions across segment boundaries.

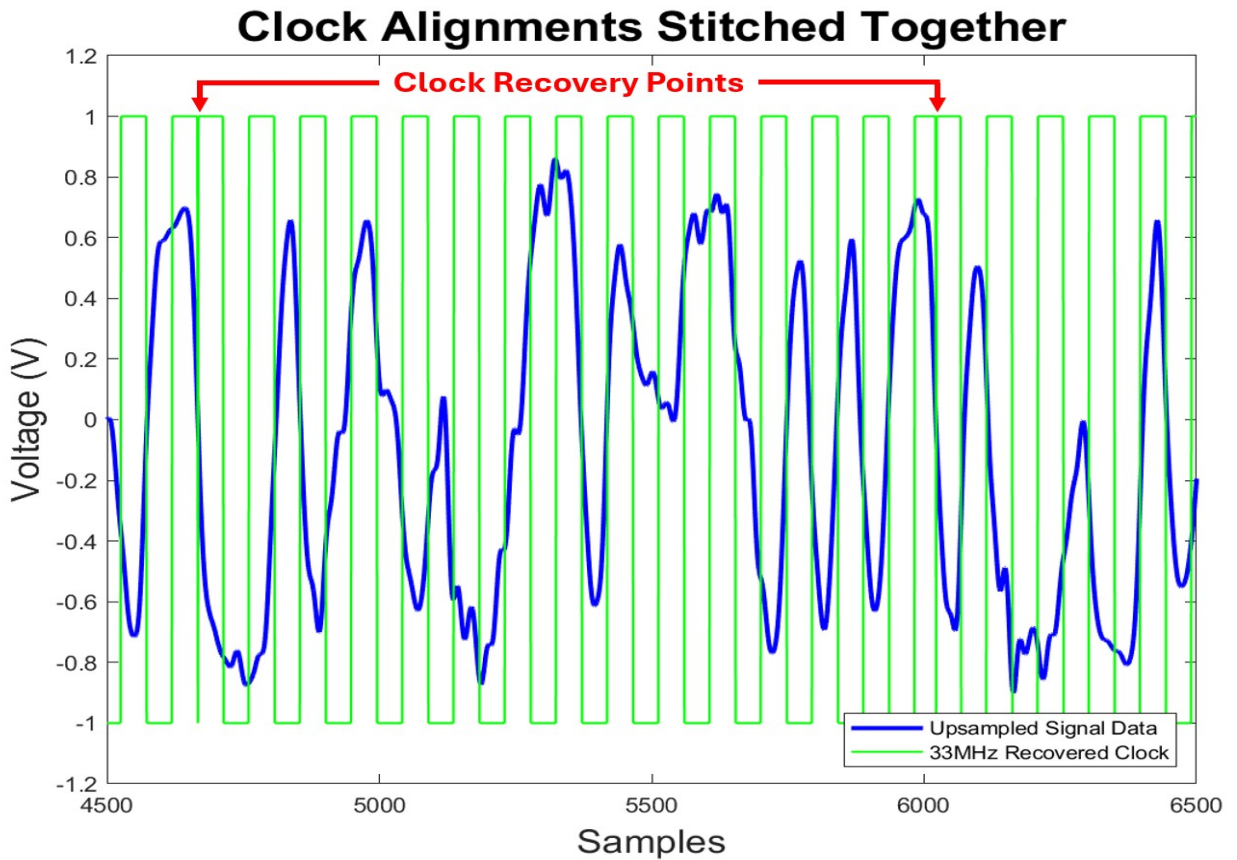


Figure 4.5: Stitched Clock Showing Recovery Points

The result is a continuously recovered clock that adapts across the signal. This clock is then used to determine the optimal sampling points for PAM3 symbol extraction in the subsequent stage.

PAM3 Symbol Extraction

Using the recovered clock, an algorithm was developed to extract PAM3 symbols from the signal. Ideally, the transmitted signal consists of three voltage levels (+1 V, 0 V, and -1 V), corresponding to PAM3 symbols of 1, 0, and -1. The optimal sampling point occurs at the center of each half cycle of the recovered clock, where the signal should be closest to one of these ideal voltage levels.

In practice, however, symbol extraction is nontrivial. Noise, inter-symbol interference, and signal distortion introduce local minima and maxima between ideal levels, making direct classification unreliable. To address this, a multi-stage decision process was developed to improve symbol accuracy.

Figure 4.6 shows the overall symbol extraction pipeline. For each symbol, a set of voting characteristics is first determined. If these characteristics provide a confident decision, the symbol is assigned directly. Otherwise, additional strategies are applied, incorporating information from neighboring symbols. This process guarantees symbol assignment, although some symbols may be flagged for later refinement.

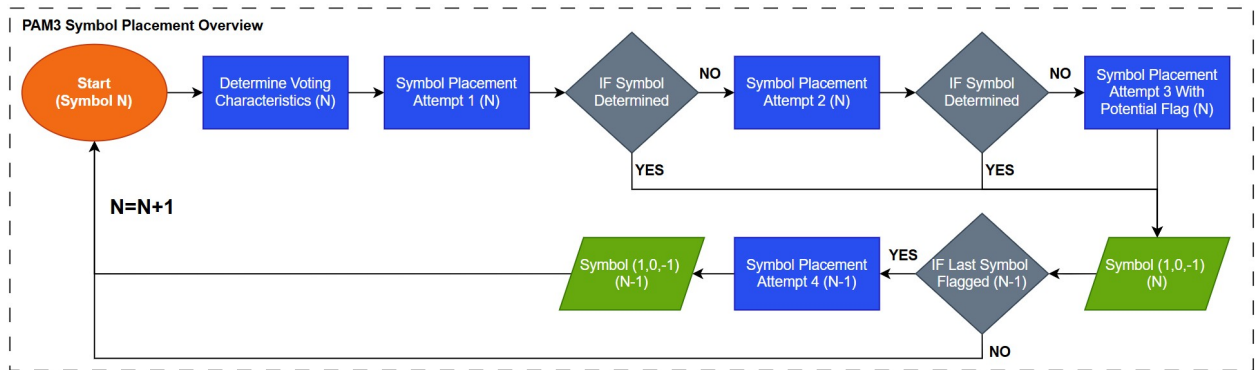


Figure 4.6: PAM3 Symbol Extraction Overview Flow Diagram

Voting Characteristics

For each half cycle of the recovered clock, all samples are extracted; however, only the middle portion of the samples is used, as it provides the most reliable representation of the symbol. Samples near the edges are more likely to be affected by transitions.

Each sample is classified into one of five regions based on voltage thresholds: High (H), High-Mid (HM), Mid (M), Low-Mid (LM), and Low (L). The number of samples in each region is counted, and a majority region is determined.

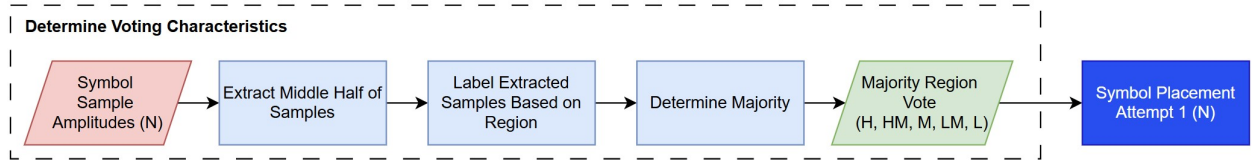


Figure 4.7: Determination of Voting Characteristics Flow Diagram

Symbol Placement Strategy

Attempt 1: Majority Vote If the majority region is H, M, or L, the symbol is directly assigned as 1, 0, or -1, respectively. These cases correspond to strong confidence decisions.

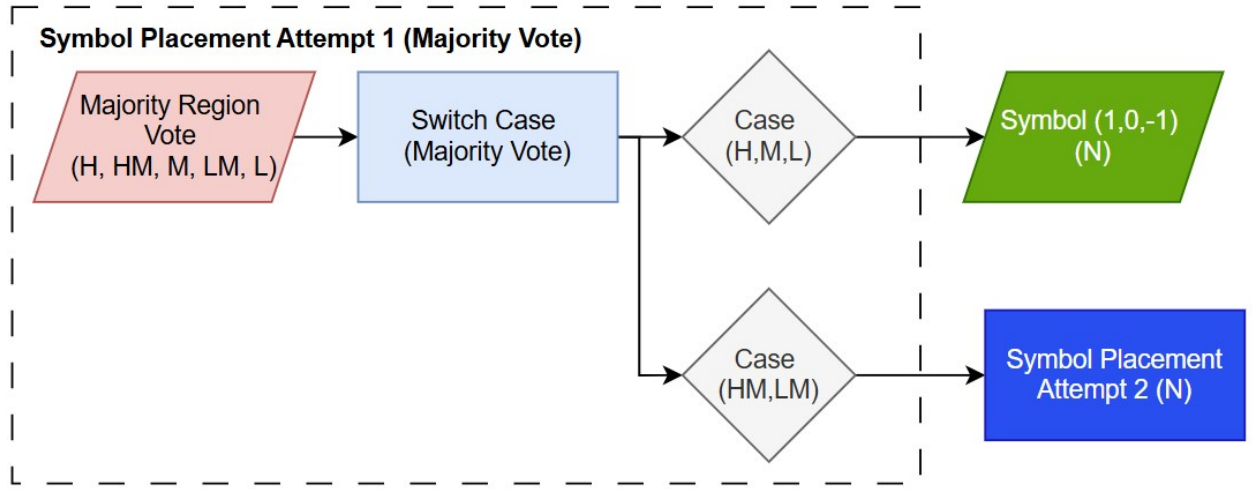


Figure 4.8: Symbol Placement Attempt 1 Flow Diagram

If the majority falls within the ambiguous regions (HM or LM), the algorithm proceeds to the next stage.

Attempt 2: Previous Symbol Comparison The center sample of the current symbol is compared with that of the previous symbol. If the difference is small, the current symbol is assumed to be the same as the previous symbol. Otherwise, further analysis is required.

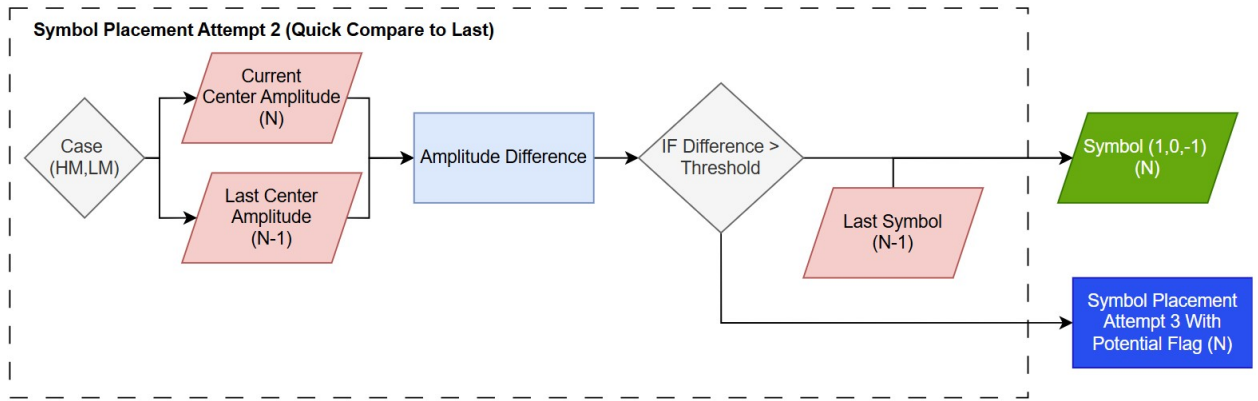


Figure 4.9: Symbol Placement Attempt 2 Flow Diagram

Attempt 3: Context-Based Decision If the first two attempts are inconclusive, a more detailed decision process is applied using both the previous symbol and the current voting characteristics.

If the previous symbol is 0, the sign of the amplitude change determines whether the current symbol is 1 or -1. If the previous symbol is ± 1 , the decision depends on both the majority region (HM or LM) and the direction of amplitude change. In ambiguous cases, the symbol is temporarily assigned and flagged for later correction.

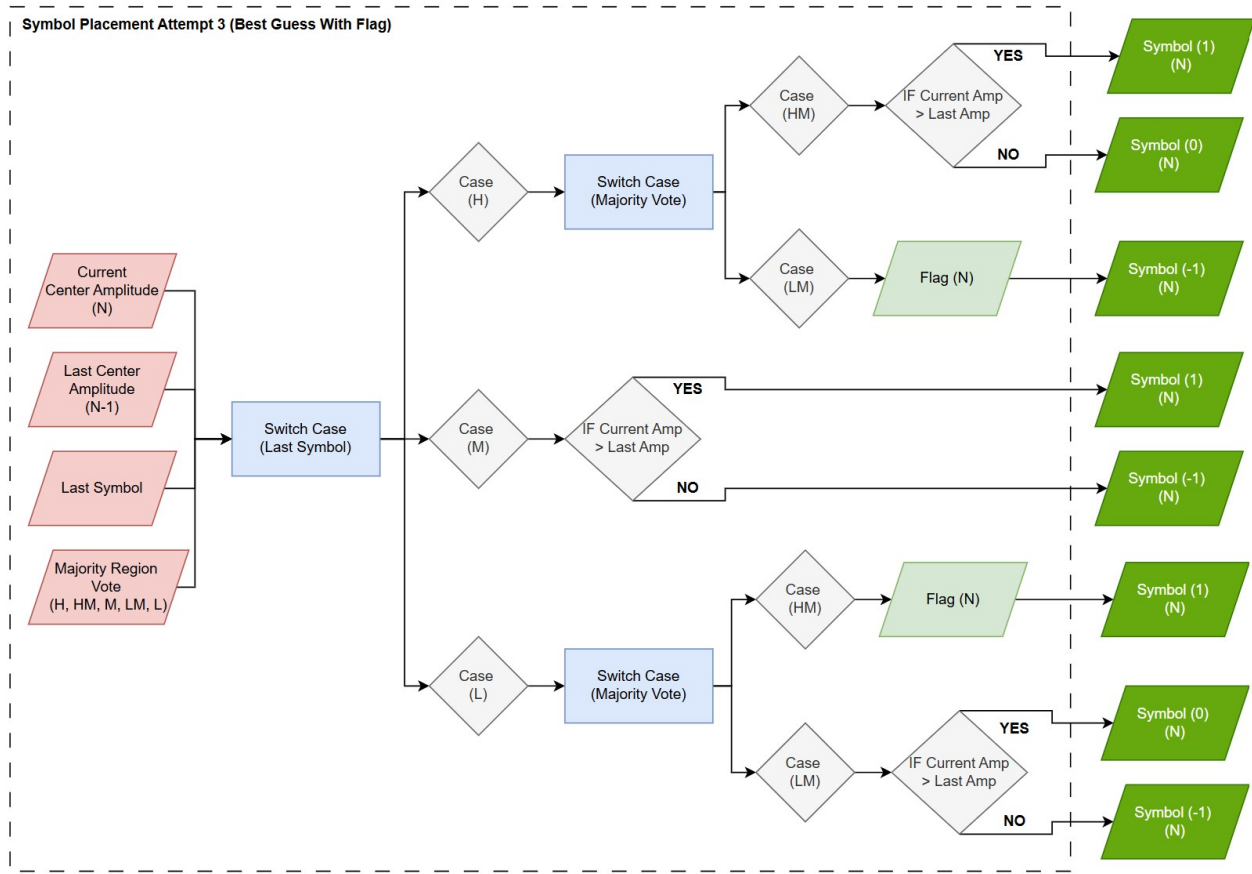


Figure 4.10: Symbol Placement Attempt 3 Flow Diagram

Attempt 4: Retroactive Correction After assigning the current symbol, the algorithm checks whether the previous symbol was flagged. If so, a three-symbol window (N-2, N-1, N) is evaluated. In cases where the surrounding symbols indicate a transition (e.g., -1 to 1 or 1 to -1), the intermediate symbol is reassigned to 0. This correction is based on observed patterns in misclassified transitions.

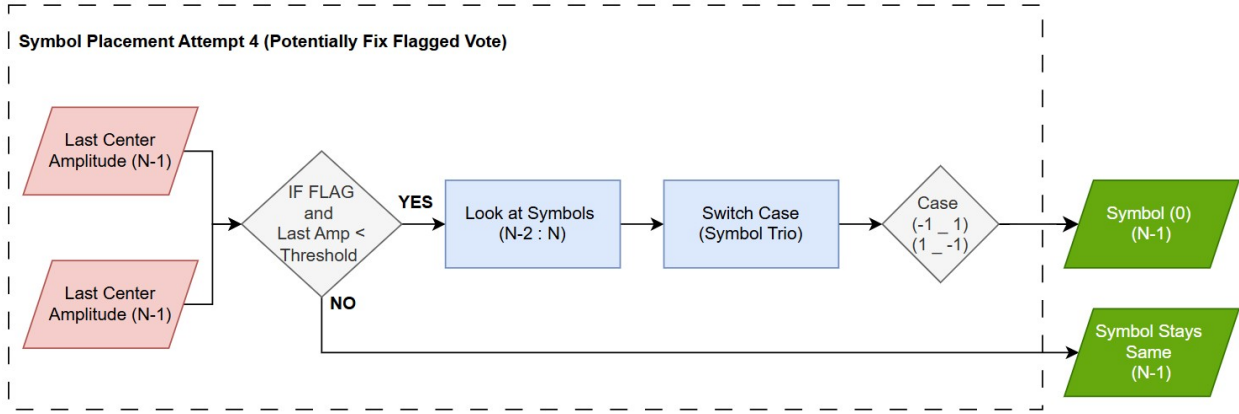


Figure 4.11: Symbol Placement Attempt 4 Flow Diagram

Summary

This multi-stage process ensures that every symbol is assigned, with increasing levels of complexity applied only when necessary. The resulting output is a discrete PAM3 symbol stream consisting of values in $\{-1, 0, 1\}$.

Prior to implementing this algorithm, decoding accuracy was approximately 10%. The introduction of this structured decision process significantly improved performance, demonstrating the importance of robust symbol extraction in the overall decoding pipeline.

Frame Extraction

As introduced in the quick search stage of the decoding pipeline, each potential frame segment contains enough samples to represent the longest possible Ethernet frame. Therefore, only a subset of the extracted PAM3 symbols corresponds to an actual frame. Frame extraction is performed by identifying the Start of Stream Delimiter (SSD) and End of Stream Delimiter (ESD) within the PAM3 stream.

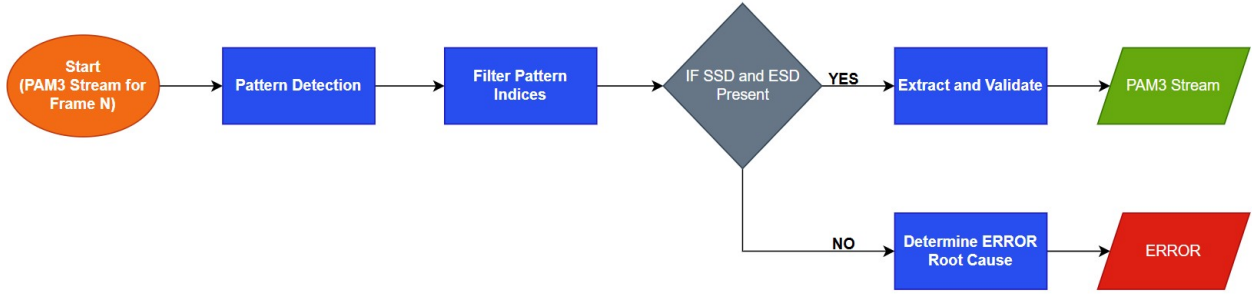


Figure 4.12: Frame Extraction Overview Flow Diagram

The extraction process begins with a pattern search to identify all candidate SSD and ESD indices within the PAM3 stream.

Significant filtering is required to remove false detections. Since the SSD should appear near the beginning of a valid frame, any SSD candidates that occur later in the segment are discarded, as they likely belong to a subsequent frame. Additionally, ESD patterns may falsely appear immediately after an SSD due to valid symbol sequences, and these cases must also be removed. A similar filtering process is applied to ESD error patterns.

In some cases, the SSD pattern may be preceded or followed by an additional zero, resulting in multiple consecutive detections of the same pattern. When this occurs, the first detected index is discarded, as it would imply an invalid symbol sequence (e.g., an extra zero following the SSD), which is not permitted under the encoding scheme.

Once valid SSD and ESD candidates are identified, the PAM3 stream is extracted from the selected SSD index to the corresponding ESD index. Because valid encoding requires symbols to be processed in pairs, the extracted sequence must contain an even number of symbols. If an odd-length sequence is obtained, the first symbol is removed to enforce proper pairing. This adjustment typically accounts for extra zeros or minor symbol classification errors. Afterward, the SSD and ESD symbols are removed, as they are not part of the frame and do not map to valid symbol conversions in subsequent stages.

Several error conditions are checked during this process. If no SSD is detected, the segment likely contains only the end of a previous frame and is labeled as an *SSD Error*. If neither an SSD nor an ESD is found, the segment is considered to contain no valid frame. If an SSD is found but no valid ESD is detected, the segment is labeled as an *ESD Error*. Additionally, if an ESD error pattern is detected, the frame is marked invalid, as indicated by the transmitter.

Finally, the presence of any invalid symbol pair, such as consecutive zeros that do not map to a valid 2T-to-3B conversion, results in immediate rejection of the frame.

The output of this stage is a paired PAM3 symbol sequence representing the extracted frame payload. If no errors are detected, this sequence proceeds to the next stage of decoding. Otherwise, the segment is returned to the clock alignment stage for reprocessing, allowing for improved timing recovery. This process repeats until a valid frame is extracted or no further improvement in clock alignment is possible.

2T to 3B Conversion

The conversion from a PAM3 symbol stream to a binary bit stream is one of the more straightforward stages in the decoding pipeline. This process is defined by the IEEE standard for Automotive Ethernet and is implemented using a fixed lookup table, shown in Table 4.1.

Table 4.1: 2T/3B Conversion Table

Scrambled Sequence	First Symbol	Second Symbol
000	-1	0
001	0	1
010	-1	1
011	0	1
100	1	0
101	0	-1
110	1	-1
111	0	-1

During transmission, the encoder groups the scrambled binary bit stream into 3-bit sequences. Each 3-bit sequence is then mapped to a pair of PAM3 symbols consisting of values $\{-1, 0, 1\}$.

Because two PAM3 symbols represent three binary bits, a mismatch can occur between the total number of transmitted bits and the required grouping into 3-bit sequences, as 8 (the number of bits in a byte) is not divisible by 3. To address this, the transmitter may append one or two *stuff bits* at the end of the bit stream to ensure that the final group forms a complete 3-bit sequence.

To reverse this process, the received PAM3 stream is first grouped into symbol pairs, beginning immediately after the SSD and continuing up to the ESD. Each symbol pair is then mapped back to its corresponding 3-bit sequence using the conversion table, resulting in a scrambled binary bit stream.

Before proceeding to the next stage, the length of the reconstructed bit stream must be validated. Specifically, the length (excluding the SSD, ESD, and the initial portion of the stream affected by the preamble) must be a multiple of 8 to form valid bytes. The first 9 bits are excluded from this check because the SSD replaces the initial 9 bits of the preamble,

which can distort alignment at the beginning of the stream.

If the resulting bit stream length is not a multiple of 8, this is attributed to the presence of appended stuff bits. In this case, one or two bits are removed from the end of the stream to restore proper byte alignment.

Descrambling

Link Negotiation

After the transmitter converts the binary sequence representing the frame to be transmitted, it scrambles that sequence. This scrambling is performed for clock recovery purposes. Most clock recovery software and circuitry rely on transitions between symbols, and increasing the number of transitions reduces the risk of clock recovery failure. Without scrambling, if a network transmits a frame containing long stretches of a single symbol, the receiver may have difficulty accurately capturing the data. A pseudo-random scrambling technique ensures that, on average, sufficient transitions occur.

This pseudo-randomization also ensures that the receiver can still decode the incoming data, as both endpoints share a recognizable pattern. To establish this pseudo-random sequence, the two endpoints perform link negotiation. During this process, the devices first determine which will act as the master and which will act as the slave. There are three transmission modes: SEND_Z, SEND_I, and SEND_N. The SEND_Z mode transmits constant 0 symbols, SEND_I is used for training the idle sequence, and SEND_N is used for transmitting actual data or idle sequences.

During the handshake, the master transitions from SEND_Z to SEND_I while the slave remains in SEND_Z. During this time, the slave synchronizes its clock and locks its scrambler. The master then remains in SEND_I while the slave transitions to SEND_I, where it trains its echo canceler. The master subsequently locks its scrambler, and both devices now agree on the scrambler state. Once the scrambler state is locked and consistent between both devices,

they enter SEND_N mode, where either endpoint can transmit frames or idle sequences between frames.

When developing a passive tap, it is assumed that the tap does not participate in the link negotiation process. The goal of the passive tap is to observe any full-duplex Automotive Ethernet signal and decode frames from either endpoint at any point in time. As a result, the passive tap cannot lock the scrambler state with the master and slave, making it difficult to directly descramble the binary sequence. Therefore, the scrambler state must be solved for on a per-frame basis. To accomplish this, a detailed understanding of the scrambling process is first required.

Scrambling Method

To begin, it is assumed that the master and slave have agreed upon a scrambler state. This scrambler state is contained within a LFSR. For Automotive Ethernet, the LFSR is 33 bits wide and is filled with a pseudo-random binary sequence. It is assumed that this register is completely filled and identical for both the master and the slave. First it will be described how a 3-bit binary sequence is scrambled based on the initial state of the LFSR, and then explain how the LFSR progresses to scramble the remaining data.

As seen in Figure 4.13, consider a 3-bit sequence that is to be transmitted and must be scrambled. For each iteration of the LFSR, 3 bits of data are scrambled. The example shown is $TX_{data}[2 : 0]$. The LFSR has an initial state agreed upon by the master and slave, where positions 0 through 32 are all populated with binary values. The LFSR can also be denoted as Scr_n . The bit-wise notation for Scr_n is reversed, with position 0 located at the far left of the register, to remain consistent with the IEEE standard.

The first step in the scrambling process is to generate a 3-bit sequence based on taps within the LFSR. This sequence is denoted as $Sy_n[2 : 0]$. Each bit of $Sy_n[2 : 0]$ is defined by an XOR operation between two tapped bits of the LFSR. These tap locations are illustrated

in Figure 4.13, with colors corresponding to their positions in the register. Specifically, $Sy_n[2]$ is the XOR of bits in positions 6 and 16 of the LFSR, $Sy_n[1]$ is the XOR of bits in positions 3 and 8, and $Sy_n[0]$ is simply equal to the bit in position 0. Once $Sy_n[2 : 0]$ is determined, the original data $TX_{data}[2 : 0]$ is XOR'd with $Sy_n[2 : 0]$ to produce the scrambled output $Sd_n[2 : 0]$.

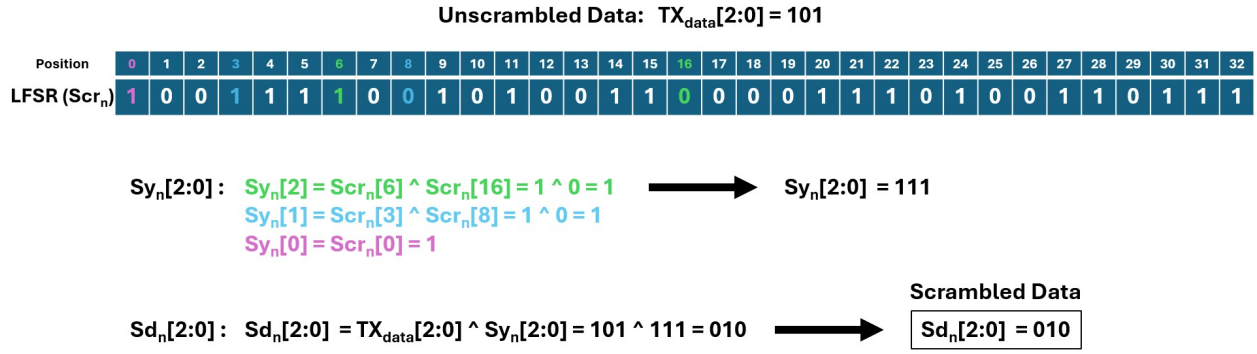


Figure 4.13: Scrambling Process for 3-Bit Sequence With Initial LFSR Filled

After scrambling 3 bits of data, the LFSR updates to process the next 3 bits. To perform this update, the LFSR computes a feedback bit based on tap positions that differ between the master and slave. The feedback bit is obtained by XOR'ing the two specified taps. For the master, these taps are positions 13 and 32, while for the slave they are positions 20 and 32. Once the feedback bit is computed, all bits in the register shift to the right. The bit in position 32 is shifted out of the register, and the newly computed feedback bit is inserted into position 0. At this point, the LFSR has been updated and is ready to scramble the next 3-bit sequence. This process repeats continuously as long as the master and slave remain linked.

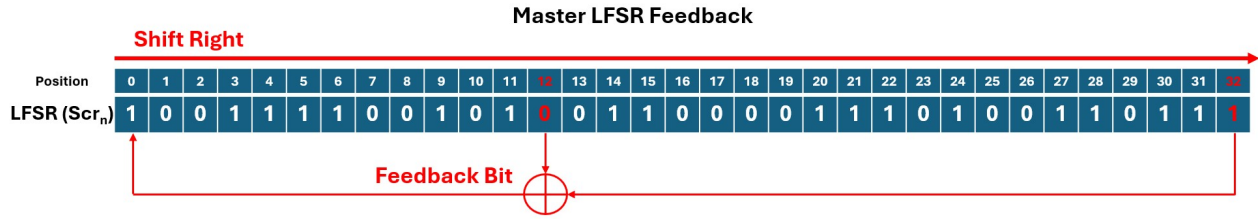


Figure 4.14: LFSR Feedback and Shift Example for Master Device

Target Sequence to Solve

Now that the scrambling process is understood, the next step is to work backwards to recover the original data from the scrambled sequence. Assuming the LFSR is fully defined, descrambling follows the same process as scrambling. This type of mechanism is known as an additive scrambler, where the scrambling and descrambling operations are identical. However, to reach this point, the LFSR state must first be solved. From the decoded binary sequences of potential frames obtained in earlier steps, the LFSR is initially unknown. Therefore, knowledge of the frame structure must be leveraged to determine the LFSR state at the start of each frame.

In particular, the preamble of an Automotive Ethernet frame is required, as it is the only portion that is consistent across all frames. The preamble consists of 8 bytes, represented in hexadecimal as 55 55 55 55 55 55 55 D5. This known sequence enables the LFSR to be solved by working backwards through the scrambling process with several key considerations.

The SSD replaces the first 9 bits of the preamble in Automotive Ethernet. This is important because it removes several 3-bit sequences that could otherwise be used to solve the LFSR. As a result, this method relies on exactly 16 iterations of 3-bit scrambled sequences to solve all 33 LFSR positions. Of the 64 known preamble bits, 9 are replaced by the SSD, leaving 55 usable bits. This provides a maximum of 18 possible 3-bit iterations, meaning that solving the LFSR in 16 iterations is sufficient but leaves little margin.

To construct the required sequence, Figure 4.15 below illustrates how the target

sequence is determined. Although the preamble is fixed, the data is not transmitted in its original order. Each byte is first reversed, such that transmission occurs from the least significant bit (LSB) to the most significant bit (MSB). After reversing the bytes, the data is concatenated into a single binary stream and divided into 3-bit sequences. These 3-bit sequences are then individually reversed, since they are transmitted in reverse order. Finally, the target sequence is extracted by removing the first 9 bits (replaced by the SSD) and selecting the next 48 bits, corresponding to 16 3-bit sequences. This resulting sequence represents TX_{data} and is used to solve for the LFSR.

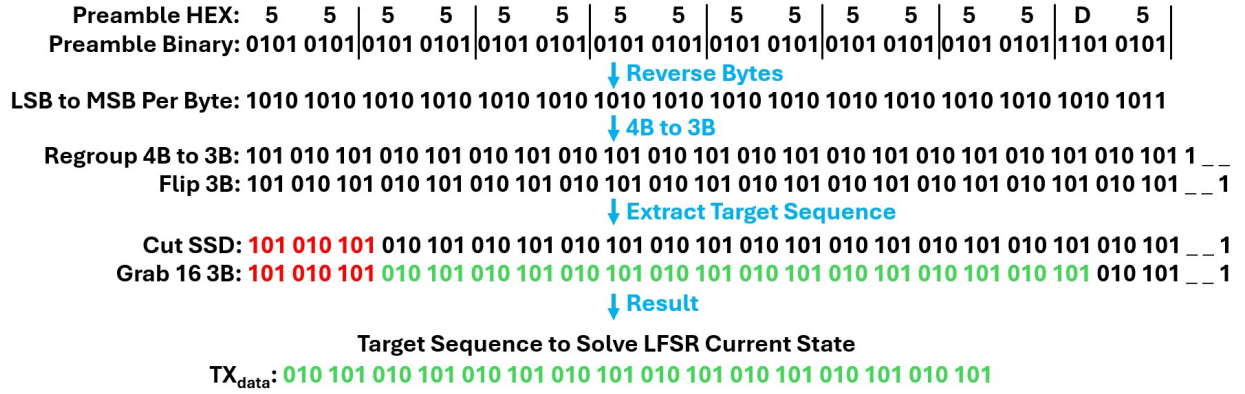


Figure 4.15: Determining Target Sequence to Solve LFSR

Solving for LFSR Current State

The LFSR can now be solved by working backwards through the scrambling process. Since the target sequence is known and the scrambled sequence has already been recovered through decoding, the LFSR state can be determined using the known scrambling equations.

The process begins with the first 3-bit sequence of TX_{data} , denoted as $TX_{\text{data}}[2 : 0]$. This sequence is XOR'd with the corresponding 3-bit sequence of the received scrambled data, $Scr_n[2 : 0]$, producing $Sy_n[2 : 0]$. Once $Sy_n[2 : 0]$ is known, the LFSR tap equations can be applied. Specifically, $Sy_n[2]$ is the XOR of $Scr_n[6]$ and $Scr_n[16]$, $Sy_n[1]$ is the XOR of $Scr_n[3]$

and $Scr_n[8]$, and $Sy_n[0]$ is equal to $Scr_n[0]$. This process repeats for each corresponding 3-bit sequence in TX_{data} and Sd_n . These relationships are illustrated in the scrambling figure.

Although this process appears straightforward, the challenge is that the LFSR tap values are initially unknown. The LFSR starts completely empty, and while $Sy_n[2 : 0]$ is known, each of its bits depends on two unknown LFSR positions (except for $Sy_n[0]$). As a result, the LFSR must be progressively populated with known values until enough positions are determined to consistently solve the tap equations.

To begin populating the LFSR, Figure 4.16 illustrates the first iteration of the solve process. Here, TX_{data} represents the target sequence, and Sd_n represents the recovered scrambled sequence from prior decoding steps. Each iteration is color-coded to show how the LFSR is gradually filled. In the first iteration (highlighted in red), $TX_{data}[2 : 0]$ and $Sd_n[2 : 0]$ are XOR'd to produce $Sy_n[2 : 0]$. At this stage, only $Scr_n[0]$ can be directly determined, since it is equal to $Sy_n[0]$. The remaining positions cannot yet be solved, although the relationship between them (same or opposite) is known from $Sy_n[2 : 1]$. This relationship is temporarily stored for $Sy_n[2]$ using variables B1 and B2, representing the corresponding LFSR positions. Storing this information is necessary, as there are not enough 3-bit sequences available to solve the LFSR without preserving these constraints.

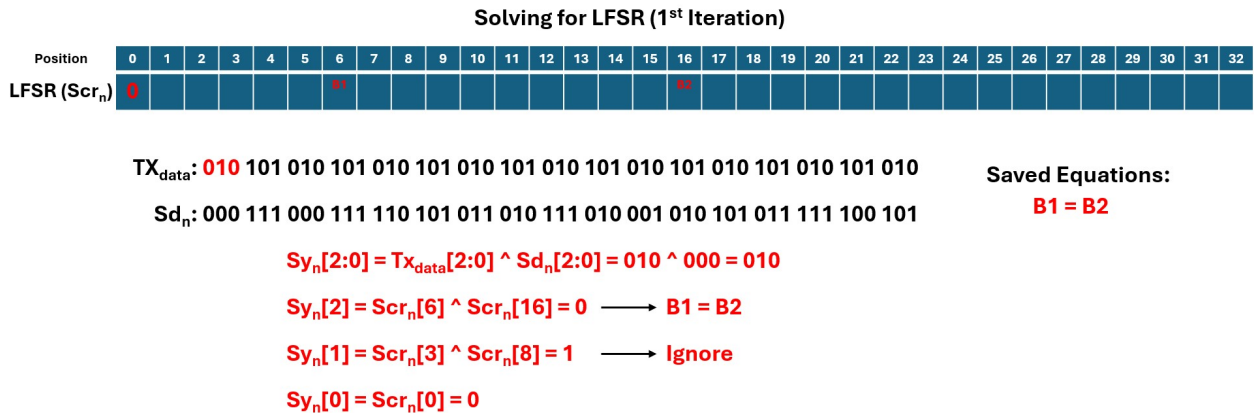


Figure 4.16: Solving LFSR 1st Iteration

After completing the first iteration, the LFSR shifts to the right, consistent with normal LFSR operation during scrambling or descrambling. This process continues, where the 0th position can always be solved and the equation for $Sy_n[2]$ is stored. Figure 4.17 below illustrates the 4th iteration, where additional LFSR positions become solvable. In particular, the equation for $Sy_n[1]$ now contains only one unknown, $Scr_n[8]$, because $Scr_n[3]$ has been determined through prior shifts. Additionally, B1 from an earlier iteration (highlighted in orange) is now known, allowing B2 to be solved using the previously stored relationship. Although this could be resolved immediately, it is deferred until all stored equations are processed together for simplicity. From this point onward, both $Scr_n[0]$ and $Scr_n[8]$ can be solved in each iteration.

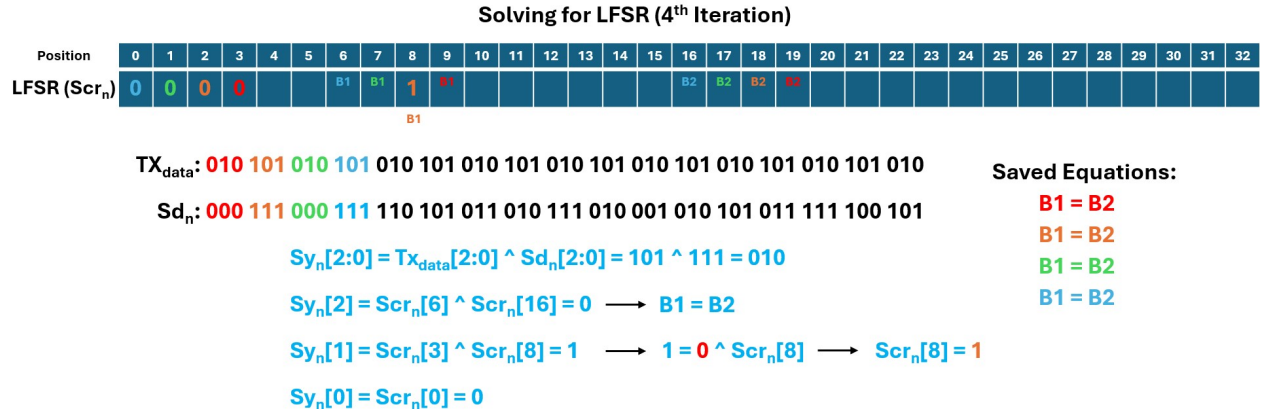


Figure 4.17: Solving LFSR 4th Iteration

As the iterations continue, the LFSR shifts right and the equations for $Sy_n[1]$ and $Sy_n[0]$ are used to solve additional positions. By the 7th iteration, more information becomes available. At this point, previously solved bits have shifted such that $Scr_n[6]$ is now known, allowing $Scr_n[16]$ to be determined using the $Sy_n[2]$ equation. Several stored B1 values are also now known, enabling their corresponding B2 values to be solved later in a batch step.

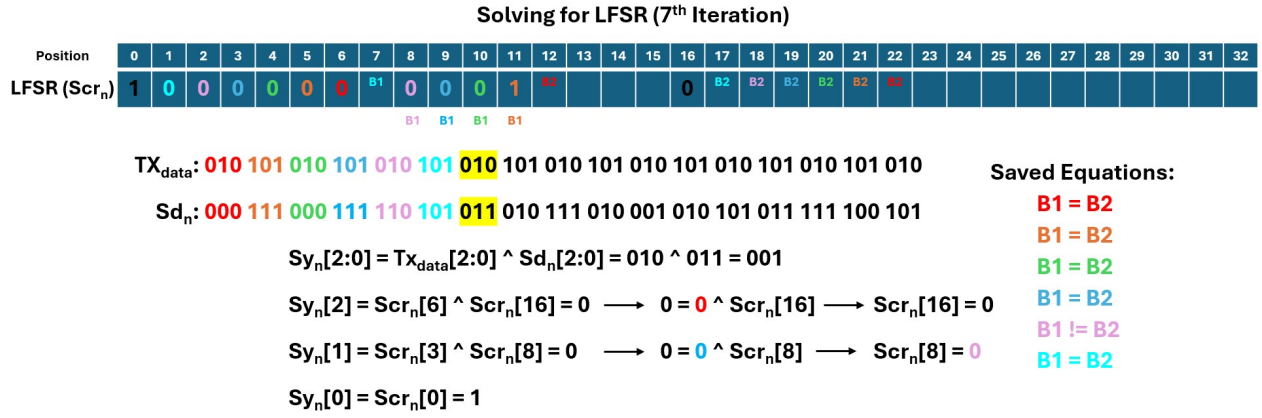


Figure 4.18: Solving LFSR 7th Iteration

The process continues in the same manner, leveraging all $Sy_n[2 : 0]$ equations to populate the LFSR. By the 12th iteration, a sufficient number of positions have been determined such that all stored B1 values correspond to known bits. The saved relationships can now be used to solve all corresponding B2 values and fill the remaining positions. At this stage, $Scr_n[0]$ through $Scr_n[27]$ have been determined. Beyond solving, the Sy_n equations also serve as an error detection mechanism. Once both inputs to an equation are known, the result can be verified against the computed Sy_n value. Any mismatch indicates an error and terminates the decoding process.

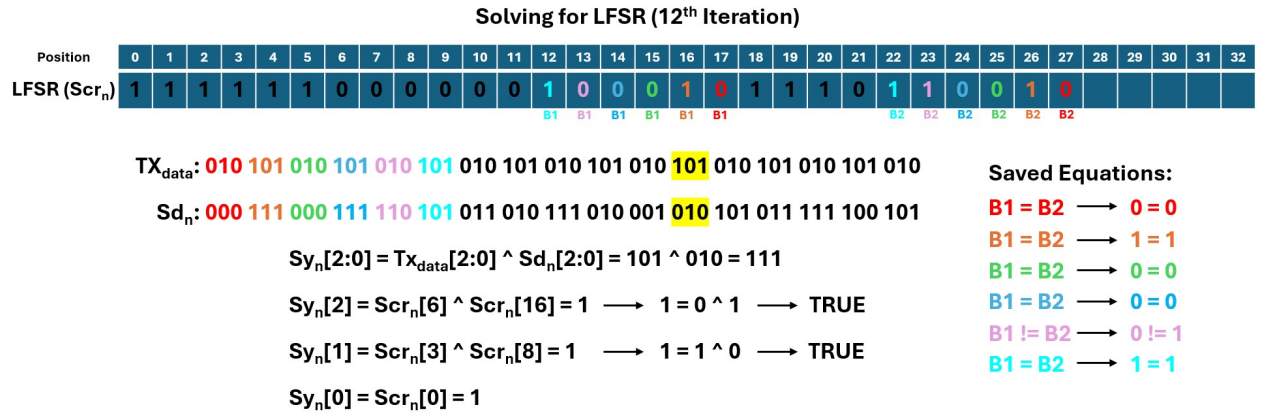


Figure 4.19: Solving LFSR 12th Iteration

After the 12th iteration, the LFSR continues to shift right, but $Sy_n[2]$ and $Sy_n[1]$ are used only for error detection rather than solving new positions. The remaining unknown bits are filled solely using $Sy_n[0]$, which directly determines $Scr_n[0]$ at each step. By the 17th iteration, all 33 positions of the LFSR have been successfully solved, assuming no errors were detected.

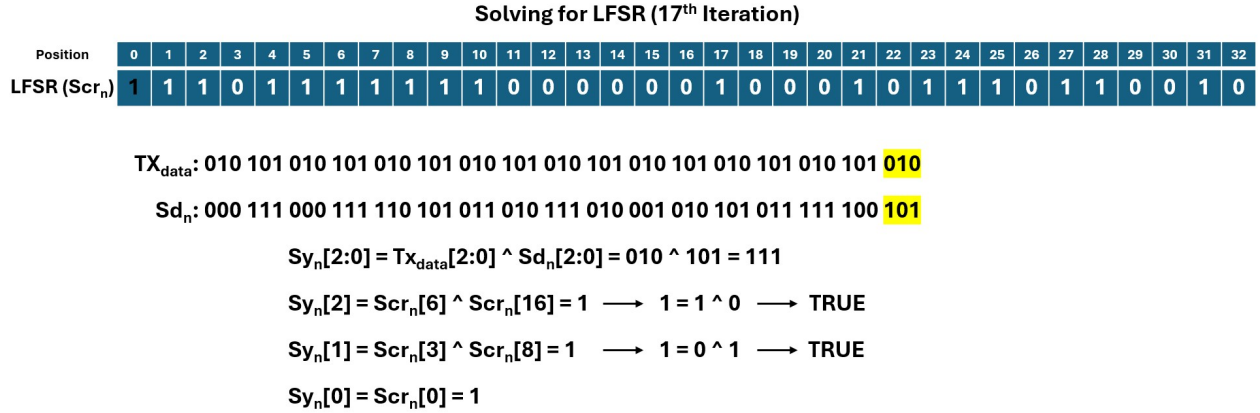


Figure 4.20: Fully Solved LFSR After 17th Iteration

Descrambling Remaining Scrambled Sequence

Once the LFSR has been fully solved, the remaining scrambled bits can be descrambled using the same process as scrambling. The only difference is that $TX_{data}[2 : 0]$ is recovered by XOR'ing the generated $Sy_n[2 : 0]$ with the received scrambled sequence $Sd_n[2 : 0]$. As shown below, the LFSR is used to compute $Sy_n[2 : 0]$, which is then used to recover the original $TX_{data}[2 : 0]$ from $Sd_n[2 : 0]$.

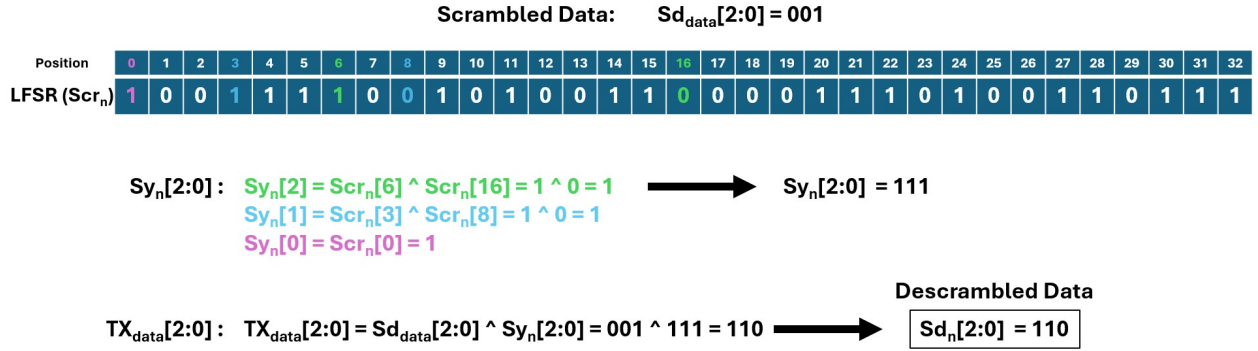


Figure 4.21: Descrambling Process

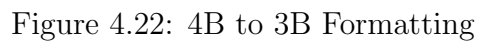
After each 3-bit sequence is processed, the LFSR computes the feedback bit and shifts to the right, inserting the feedback bit into the 0th position. This process repeats, with each LFSR update producing 3 additional descrambled bits, until the entire sequence has been processed.

Since the preamble was used to solve the LFSR but is still part of the frame, the target sequence used during the solve step must be concatenated with the remaining descrambled data to reconstruct the full frame. Additionally, because the SSD replaces part of the preamble, the expected preamble bits that were overwritten must be reinserted to complete the frame.

Reorganize Bytes

Once the full descrambled binary sequence representing the frame has been reconstructed, the data must be reorganized into a byte-aligned format with the MSB first in each byte. This ensures that, when converted to hexadecimal, the frame appears in its expected format.

This step reverses the 4B/3B transformation described earlier during target sequence construction. Specifically, the descrambled data must be converted from 3-bit groupings back to 4-bit groupings. This is done by first reversing each 3-bit sequence, then concatenating



This entire descrambling procedure must be performed for each individual frame, as gaps in transmission disrupt the LFSR state. Despite this, the computational cost is low, as the process primarily consists of simple bitwise operations and register shifts. This approach enables descrambling without participation in link negotiation, allowing frames from either the master or slave device to be recovered. The final result is the binary sequence corresponding to the transmitted frame.

The final step in the decoding process is determining whether the frame is valid. This is accomplished by performing a CRC on the binary sequence representing the frame. A CRC involves computing a value from the frame data and comparing it to the final field of the

frame, known as the Frame Check Sequence (FCS). If the computed value matches the FCS, the frame is considered valid; otherwise, it is deemed invalid. The CRC procedure itself is consistent across all frames.

Before performing the CRC, the data must be properly organized. The FCS consists of 4 bytes located at the end of the frame, so it is first removed and set aside for later comparison. The CRC is computed over the remainder of the frame, excluding the preamble, which is also temporarily removed. This leaves a binary sequence starting from the MAC Destination field through the end of the data field. Additionally, each byte is flipped from LSB first to MSB last.

Once the data is prepared, the CRC calculation can begin. The first 32 bits of the reorganized frame sequence are complemented (i.e., bitwise inverted). Then, 32 zero bits are appended to the end of the binary sequence. This augmented sequence, denoted as $M(x)$, is divided by a generator polynomial defined by the IEEE Ethernet standard. The polynomial is:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

This polynomial corresponds to the following binary sequence, where each exponent indicates the presence of a 1 at that position:

$$G(x) = [100000100110000010001110110110111]$$

The division of $M(x)$ by $G(x)$ is performed using modulo-2 arithmetic. Conceptually, this involves aligning $G(x)$ with the most significant 1 in $M(x)$ and performing a bitwise XOR. The polynomial is then shifted and XOR'd with $M(x)$ at each position where a leading 1 exists. This process continues until the LSB of $G(x)$ aligns with the LSB of $M(x)$. The

result of this division is called $R(x)$, which consists of the last 32 bits of $M(x)$ after all XOR operations. All preceding bits in $M(x)$ should have been reduced to zero, leaving $R(x)$ as the remainder of the division.

Finally, $R(x)$ is reorganized into bytes, flipping each byte back from LSB to MSB. This resulting value is the computed FCS, which is then compared to the original FCS extracted from the frame. If they match exactly, the frame is deemed valid; otherwise, it is considered invalid. CRC is a widely used method for error detection in Ethernet and other transceivers.

Results

Testing Methodology

The passive tap machine learning approach used the first of ten experimental captures for model training, while the remaining nine captures were reserved for testing. To enable a direct comparison, these same nine captures were also used to evaluate the decoding algorithm. Rather than operating on predicted component signals, the decoding algorithm was applied to ground truth data.

Following the main loop described in the decoding overview (Figure 4.1), all potential frames and alignment markers were extracted from each dataset, producing a set of candidate frames for decoding. Each potential frame was processed sequentially through the decoding pipeline, including clock recovery, symbol extraction, descrambling, and CRC validation. If a decoding attempt resulted in an error or invalid frame, the process returned to the clock recovery stage and retried with a different alignment. This continued until no further clock recovery attempts were possible. As a result, each potential frame was ultimately classified as either a valid frame, an invalid frame, or as not containing a frame.

For frames classified as invalid, the cause of failure was recorded. Two primary error types were identified: PAM3 errors and SSD/ESD errors. PAM3 errors were the most common and occurred when an incorrectly decoded PAM3 symbol prevented successful frame

reconstruction. These errors could manifest at multiple stages, including frame extraction, 2T-to-3B conversion, descrambling, or CRC validation. In all cases, the root cause was an incorrect symbol decision.

SSD/ESD errors occurred under two conditions. The first was when a symbol error corrupted the Start-of-Stream Delimiter (SSD) or End-of-Stream Delimiter (ESD), preventing proper frame detection. The second, and more common, cause was due to gaps between recorded data buffers in the experimental captures. In these cases, a frame could begin or end within a gap where no data was collected, resulting in incomplete frame detection.

Decoding accuracy was defined as the ratio of successfully decoded valid frames to the total number of frames that could be decoded, excluding candidate segments determined not to contain a frame. While accuracy could also be measured at the symbol level, frame-level accuracy is the more meaningful metric, as successful communication depends on complete frame recovery. In a typical commercial transceiver with an established link, frame decoding accuracy is expected to exceed 99.9%. However, due to the need for a high pass filter (HPF) to compensate for Picoscope measurement limitations, a reduction in accuracy was anticipated. The extent of this degradation was evaluated experimentally. Notably, the Picoscope’s native reconstruction algorithm performed poorly without HPF post-processing.

Candidate segments classified as not containing a frame were primarily identified through a conservative quick-search function, which selected regions containing sequences of five consecutive zeros. This approach intentionally over-selected candidates, often identifying ESDs as potential frame starts. The decoding pipeline included validation checks to address this, ensuring that if an ESD appeared at the beginning of a segment and did not reoccur, the segment was correctly classified as not containing a valid frame. These segments were excluded from the accuracy calculation. To validate this approach, clock alignment was also performed across entire datasets without the quick-search step, yielding nearly identical

results and confirming the correctness of the classification.

HPF Tuning

Since the HPF was applied to the ground truth data, the required level of filtering was initially unknown. To determine the optimal setting, a sweep was performed in which progressively more low-frequency components were removed, and the resulting decode accuracy was evaluated at each step.

Table 4.2 summarizes the results of this sweep. For each HPF setting, the number of low-frequency components removed in the frequency domain is shown alongside the total number of valid frames, total frames evaluated, and overall decoding accuracy. Additionally, the number of PAM3 errors, SSD/ESD errors, and instances where no frame was found are reported to provide further insight into failure modes.

Table 4.2: HPF Sweep Accuracy Results

HPF	Accuracy	VALID Frames	Total Frames	PAM3 Errors	SSD/ESD Errors	No Frames Found
0	78.1%	589	754	119	46	583
25	87.5%	650	743	62	31	285
50	92.8%	683	736	29	24	188
75	93.3%	685	734	26	23	150
100	93.0%	682	734	28	24	149
125	93.2%	684	734	26	24	148
150	92.8%	681	734	29	24	150
175	92.1%	677	735	33	25	162
200	90.8%	669	737	43	25	179
225	90.0%	662	736	48	26	202
250	89.6%	661	738	50	27	217
275	87.4%	647	740	64	29	236
300	85.8%	636	741	74	31	268

The results show a clear improvement in decoding accuracy as low-frequency components are initially removed. Accuracy increases significantly from 78.1% with no filtering to over 92% once approximately 50 low-frequency components are removed. This indicates that low-frequency noise or drift in the captured signals has a strong negative impact on symbol recovery and overall decoding performance.

The optimal performance occurs in the range of 50 to 150 removed frequency components, where the decoding accuracy remains relatively stable between 92.8% and 93.3%. Within this region, PAM3 errors are minimized, suggesting that the LPF effectively improves symbol detection without distorting the signal.

Beyond this range, performance begins to degrade as more aggressive filtering is applied. From 175 onward, a gradual decline in accuracy is observed, accompanied by an increase in PAM3 errors. This indicates that excessive removal of low-frequency content begins to distort the underlying signal, negatively impacting symbol reconstruction.

Interestingly, SSD/ESD errors remain relatively stable across all HPF settings, suggesting that these errors are less sensitive to frequency-domain filtering and are more likely driven by frame boundary conditions or data gaps rather than signal quality.

Overall, this sweep demonstrates that moderate high-frequency filtering is critical for achieving high decoding accuracy, while overly aggressive filtering introduces signal distortion that outweighs its benefits. Based on these results, an HPF setting in the range of 75 to 125 provides a strong balance between noise reduction and signal integrity.

Final Results and Impact

Based on the HPF sweep, the range of 75 to 125 removed low-frequency components was identified as optimal. Selecting the midpoint of this range, an HPF setting of 100 was chosen as the final configuration and will be used for all model training in the passive tap implementation.

Table 5.8 presents the decoding results for this HPF setting, including a breakdown by component signals T_A and T_B to evaluate any differences in performance.

Table 4.3: HPF Sweep Accuracy Results

Signal	Accuracy	VALID Frames	Total Frames	PAM3 Errors	SSD/ESD Errors	No Frames Found
T_A	95.7%	358	374	7	9	69
T_B	90.0%	324	360	21	15	80
Combined	93.0%	682	734	28	24	149

A noticeable performance gap of approximately 5% exists between T_A and T_B . This discrepancy is most likely due to differences in signal quality during acquisition. In particular, the Picoscope appears to have more difficulty accurately capturing T_B , which may be attributed to probe variation, slight placement inconsistencies, or channel-specific hardware effects. While the exact cause is not definitively known, the elevated PAM3 error count for T_B supports the conclusion that symbol reconstruction is less reliable for this component.

Despite this imbalance, the overall decoding accuracy of 93.0% represents a substantial improvement over the baseline Picoscope decoding performance. This indicates that the developed decoding algorithm is effective at recovering valid frames when sufficient signal quality is present. Furthermore, SSD/ESD-related errors are largely driven by dataset limitations, specifically gaps between captured buffers where partial frames may be lost. If these cases are excluded, the effective decoding accuracy increases further, reinforcing the robustness of the approach under ideal conditions.

Overall, the decoding algorithm demonstrates strong performance in accurately recovering frames that are fully contained within the captured signal. This level of accuracy establishes a reliable ground truth baseline, enabling meaningful comparison with the machine learning-based passive tap approach. As a result, the effectiveness of the machine learning model in separating and reconstructing component signals can be directly evaluated

against a high-confidence reference.

PASSIVE DOUBLE TAP

Introduction

This chapter presents the development and evaluation of a passive double tap system for Automotive Ethernet using a RNN. The full-duplex signal separation problem is addressed by training a BiLSTM model to reconstruct individual component signals from passively tapped full-duplex measurements. The following sections describe the data preprocessing pipeline, model architecture, training procedure, hyperparameter tuning process, and final decoding performance across a range of model configurations and tap point combinations.

Preparing Data for Training

Before utilizing an RNN, the data must be preprocessed into a format compatible with the model. A preprocessing function was developed to perform this task, with behavior configurable through user-defined settings.

The function reads a specified subset of data files from a given directory, where each file is assumed to contain the signals obtained from the experimental setup (A , B , D , T_A , T_B).

To ensure proper alignment, any samples corresponding to missing values caused by gaps between recorded buffers are removed across all four signals. If the indices of missing samples are inconsistent between the component signals and full-duplex signals, the function halts with an error to prevent misaligned data from being processed.

Once aligned, the component signals are filtered to remove low-frequency drift present in the captured data. This is performed by transforming the signals into the frequency domain using the Fast Fourier Transform (FFT), zeroing a specified number of low-frequency components, and reconstructing the signals using the inverse FFT. This approach acts as a

simple HPF to mitigate baseline drift:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \quad (5.1)$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn} \quad (5.2)$$

After filtering, all five signals are combined into a single aligned dataset, which is then segmented into fixed-length sequences defined by the user, with only complete segments retained. Each segment represents an individual sample for training or evaluation using the LSTM model.

The segments are partitioned into training and testing datasets based on a user-defined ratio. During model development, the training set is used for learning, while the testing set is used for validation. When applying a trained model for prediction, all segments are treated as testing data.

To support experimentation across multiple configurations, the function allows flexible selection of inputs and outputs. The predictors consist of the full-duplex signals, which may include any combination of A , B , and D , or individual signals. The targets consist of the component signals, which include both T_A and T_B . The selected predictors and targets are extracted from each segment and stored in cell arrays.

The output of this function consists of preprocessed training and testing predictor and target datasets, providing a complete pipeline from raw signal captures to structured inputs suitable for supervised machine learning.

Model Training

After preprocessing, the data is formatted for model training and fed into an RNN defined using MATLAB's built-in deep learning framework. A function was developed

to configure and train a multivariate time-series regression model based on user-defined parameters.

The function accepts the preprocessed training inputs and targets along with several configuration parameters: the number of input variables, the number of output variables, the hidden layer size, the number of training epochs, and the mini-batch size.

The dimensionality of the network is automatically determined based on the selected predictors and targets. The input layer contains one, two, or three features depending on the combination of full-duplex signals used, while the output layer always contains two dimensions corresponding to the two component signals.

The network is constructed as a deep sequential model beginning with a sequence input layer, which defines the structure of the incoming data and serves as the entry point to the network. The input size is determined by the number of selected predictors and the segment length, forming an input matrix of size $features \times sequence\ length$.

This is followed by an RNN layer, which serves as the core recurrent component. The number of hidden units, defined by the user, determines the number of memory cells used to capture temporal dependencies. As discussed in the background, the LSTM includes internal gating mechanisms-forget, input, and output gates-that regulate information flow. The output mode is set to *sequence*, meaning that predictions are generated at every time step, forming a sequence-to-sequence model. If a Bi-LSTM is trained, because the network processes data in both the forward and backward directions, the output dimensionality of this layer is twice the number of hidden units. For the LSTM and GRU models, the dimensionality remains the same as the number of hidden units.

The RNN output is then passed to a fully connected layer, which maps the high-dimensional representation to the desired output dimensionality through a linear transformation to produce the predicted component signals.

Finally, a regression layer defines the training objective using mean squared error

(MSE) as the loss function. MSE quantifies the difference between predicted outputs and ground-truth targets, enabling gradient-based optimization through backpropagation, which iteratively updates network weights to minimize prediction error.

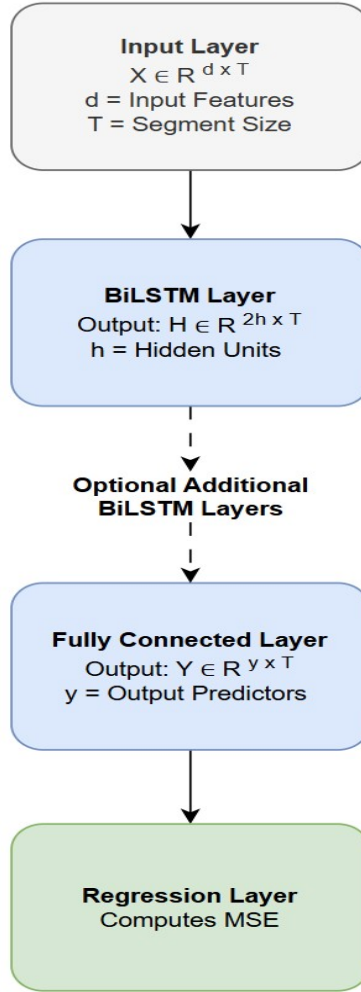


Figure 5.1: Model Architecture

With the architecture defined, training options are then specified. The optimizer is set to Adaptive Moment Estimation (Adam), a stochastic gradient-based method widely used in deep learning. Adam combines momentum-based updates with adaptive learning rates, offering faster convergence, improved stability for noisy gradients, and reduced sensitivity to hyperparameter selection—properties that make it well suited for recurrent neural network

training.

The maximum number of training epochs is user-defined and specifies how many complete passes are made through the training dataset. Increasing the number of epochs allows the model to learn more complex patterns but introduces the risk of overfitting; the optimal value is typically determined by monitoring validation performance. Early stopping criteria was implemented to terminate training when validation performance ceased to improve and had stabilized.

The mini-batch size controls how many training sequences are processed before updating network weights. Smaller batch sizes can improve generalization but introduce noisier gradient estimates and slower computation, while larger batch sizes provide smoother gradients at the cost of increased memory usage.

The number of hidden units in the BiLSTM layer is also configurable. A larger hidden unit count allows the model to capture more complex temporal relationships but increases computational cost and the risk of overfitting.

Additional training settings include data shuffling, whereby the order of training sequences is randomized at the start of each epoch. This prevents the model from learning spurious patterns attributable to data ordering and improves generalization.

Training is performed on either a CPU or GPU depending on available hardware, with GPU acceleration significantly reducing training time by parallelizing matrix operations, recurrent computations, and gradient propagation.

Throughout training, MATLAB provides a progress plot that visualizes key metrics such as loss over time, enabling monitoring of convergence behavior, detection of instability, and identification of overfitting. This visualization proved valuable in refining model configurations.

Once training is complete, the function outputs a trained RNN model ready for prediction.

Model Predictions

Evaluation Metric

To evaluate prediction performance, the mean absolute error (MAE) was used to compare the predicted component signals produced by the RNN models against the ground-truth component signals obtained by the Picoscope.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (5.3)$$

MAE is an appropriate evaluation metric for this task because it involves reconstructing continuous-valued PAM3 waveforms. MAE measures the average absolute amplitude difference between the predicted and ground-truth signals at each time sample, providing a direct assessment of waveform fidelity. Performance in this context depends on minimizing linear deviations in signal levels rather than heavily penalizing occasional large errors.

Overlapping Predictions

After the model has been trained, new data can be used to convert full-duplex signals into their component signals. A function was designed to perform the sequential prediction, reconstruction, and performance evaluation of the trained RNN model on the testing dataset. The segmented test inputs are first recombined into a single continuous time series, which is then re-windowed into overlapping segments of a fixed, user-defined length. Overlapping windows allow the model to generate predictions on sliding temporal intervals rather than on independent, non-overlapping blocks.

Overlapping is necessary due to an important observation: MAE is notably worse at the beginning and end of each segment. The function supports overlapping predictions using simple rectangular window types. RNNs require temporal context from time steps preceding—and, depending on the model type, following—the current prediction step. This

context is absent at the boundaries of a segment fed into the model, leading to elevated MAE at those locations. this observation was made on a Bi-LSTM model, with LSTM and GRU models only performing worse at the start of the predictions given they only rely on past temporal context.

The observation that motivated the overlapping strategy is illustrated Figure 5.2. On a per-sample basis, the MAE for each sample within its respective window was averaged. These per-sample MAE averages were then grouped into bins spanning 1% of the segment length in order to present the results more clearly.

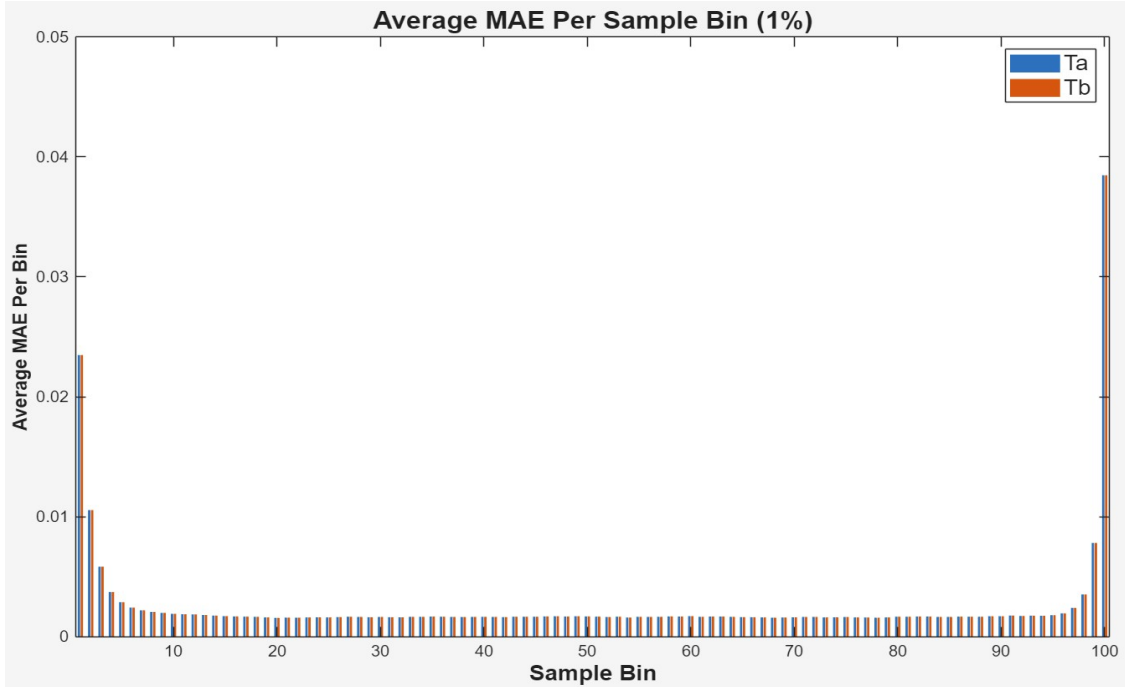


Figure 5.2: MAE Before Overlapping Predictions

The MAE is clearly elevated at both the beginning and end of each segment. It decreases rapidly from the first 1% bin, leveling off around the 10% mark, and then rises again exponentially beyond the 90% mark toward the end of the segment. This trend is consistent across both component signals. In the well-performing interior of the segment, the MAE falls in the 10^{-3} range, while the worst-performing boundary regions reach the

10^{-2} range. This difference is sufficiently large to motivate the adoption of an overlapping prediction strategy to improve signal reconstruction accuracy and decoding performance.

After a 5% overlap is applied, Figure 5.3 shows the same binned MAE calculation. This overlap was implemented using rectangular windows, extracting the middle 90% of each prediction and stitching the extracted portions together.

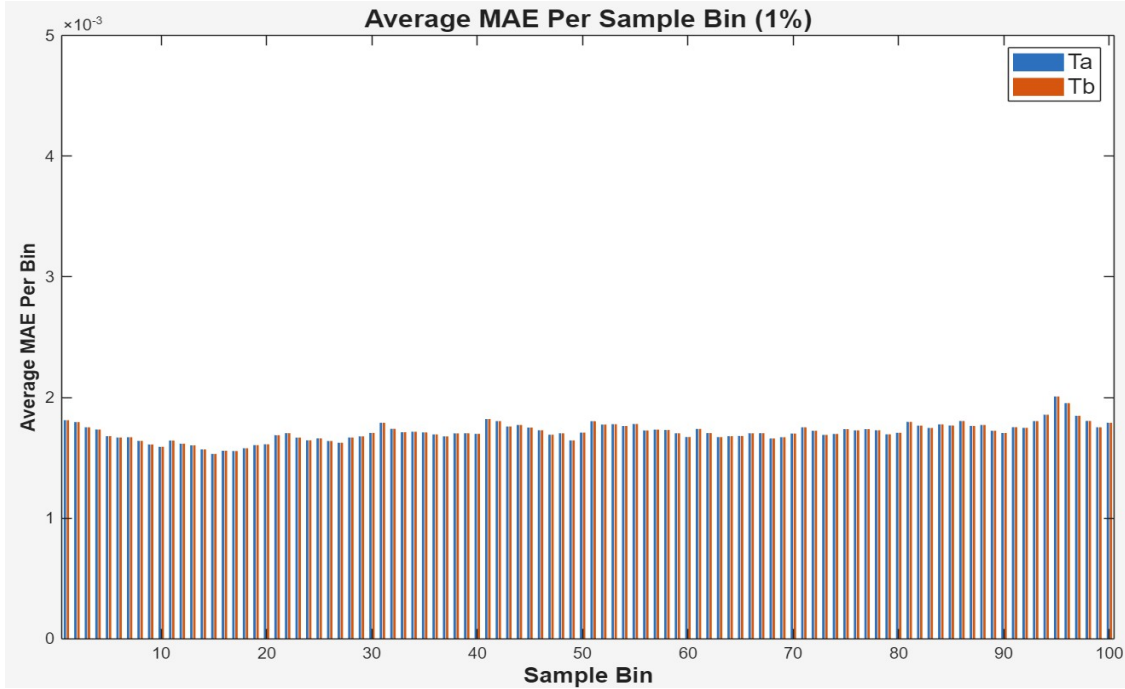


Figure 5.3: MAE After Overlapping Predictions

The 5% overlap yields substantially improved MAE results, with the error remaining near 0.002 throughout the entire segment. When binned, the MAE values for both T_A and T_B appear nearly identical. Given the magnitude of the improvement, overlapping predictions represent a necessary addition to the pipeline for achieving accurate component signal reconstruction.

One notable observation is shown in Figure 5.4, which plots the average MAE at every sample over a 1000-sample segment. This finer-resolution calculation provides additional insight into the per-signal difference in MAE, as well as a pronounced spike near sample 950.

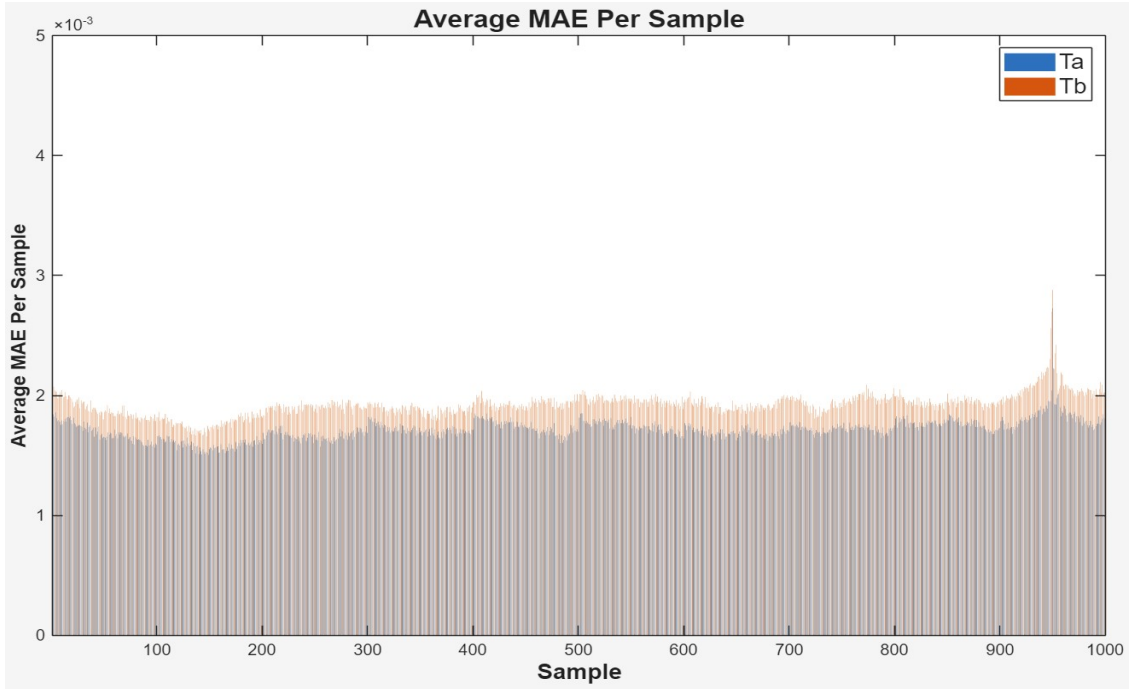


Figure 5.4: MAE After Overlapping Predictions Per Sample

Due to the rectangular windowing employed by the overlapping prediction scheme, the stitch points exhibit a noticeable spike in MAE relative to the surrounding samples. When concatenating rectangular windows, discontinuities at the boundaries can introduce localized prediction errors precisely at the stitch point.

After prediction, the overlapping samples at the start of each window are discarded to prevent duplicate estimates, and the remaining outputs are concatenated to reconstruct a continuous predicted signal. The reconstructed signal is then divided into fixed-length segments to match the original segmentation structure used for evaluation.

Comparing to Ground Truth

To enable a direct comparison with the ground-truth targets, the testing targets are processed using the same trimming, concatenation, and re-segmentation procedure applied to the predictions, ensuring temporal alignment and identical segment lengths. The function

then computes the MAE independently for both component signals over every segment, and also computes an overall MAE across the entire testing set. These metrics serve as quantitative measures of the accuracy with which the component signals are reconstructed from the full-duplex signals. The true accuracy is ultimately determined by the decoder; however, this metric provides insight into how reliably the data can be decoded.

This function returns the segmented predictions, the processed targets, and the overall MAE values for the available outputs, completing the post-processing pipeline for the sequential RNN predictions.

Determining RNN Architecture

The appropriate type of RNN model first needed to be determined. MATLAB provides built-in training architectures for GRU, LSTM, and BiLSTM models. All three were trained using the same baseline hyperparameter values and stopping criteria, and the best-performing model was selected as the reference for subsequent hyperparameter tuning.

Table 5.1: Reference Model Hyper-Parameters

Number of LSTM Cells	Number of Hidden Units	Batch Size	Segment Size	Overlap %
1	64	16	1000	10%

Table 5.2: Model Type

Model Type	Average Overall MAE	T_A MAE	T_B Average MAE
LSTM	0.0099	0.0092	0.0113
GRU	0.0116	0.0107	0.0124
BiLSTM	0.0026	0.0025	0.0027

The BiLSTM model achieved by far the best performance in predicting the component signals, with an MAE substantially lower than those of the LSTM and GRU models. This result motivated the selection of the BiLSTM as the reference architecture, with the hyperparameters listed above serving as the baseline for subsequent sweeps.

The resulting predictions are shown in the Figure 5.5. Each quadrant shows a segment of data with the predicted and ground truth waveforms plotted over one another. The predicted waveforms are shown in red, and the ground truth waveforms are shown in green. The left hand side shows two plots of segments for signal T_A that had the best average MAE on the top left and the worst on the bottom right. Same goes for T_B on the right hand side. The best segments of the prediction perform a nearly identical match to the ground truth with as low as a 0.0002 MAE. There are sections as seen in the worst segments that the MAE increases, however, the predicted waveform still follows the pattern of the ground truth waveform with slight amplitude differences.

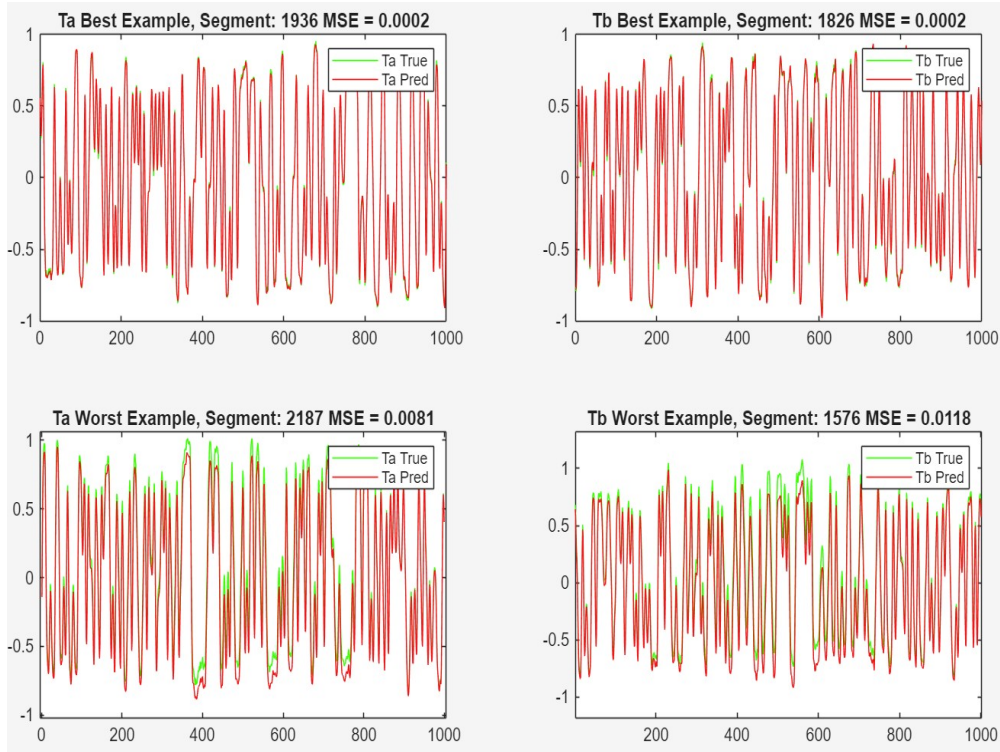


Figure 5.5: Reference Model Prediction Results

Tuning Hyper-Parameters

Bi-LSTM Layers

The reference model served as the baseline for hyperparameter tuning. For each parameter, multiple models with variations in that parameter were trained while all other parameters were held fixed at their reference values. This approach allows the effect of each parameter on model performance to be isolated.

All models employed the same early stopping criterion based on validation performance. During training, the validation root mean squared error (RMSE) was evaluated after a fixed number of training iterations. If the RMSE did not improve meaningfully between evaluations, a counter tracked the number of consecutive non-improving checks. A validation patience threshold was defined such that once the counter reached this threshold, training

was terminated early. This ensured that models trained only while meaningful improvement was occurring. As a result, individual models trained for varying numbers of epochs, but the stopping criterion was held identical across all configurations.

To begin tuning, the model architecture was modified. Rather than using a single BiLSTM layer, the output of one BiLSTM layer can be passed as input to additional BiLSTM layers, enabling the network to learn more complex temporal representations. The tradeoff is increased training and inference time due to higher model complexity. The reference model contained a single BiLSTM layer; additional models were trained with two to five total BiLSTM layers.

Table 5.3: Hyperparameter Sweep — Architecture

Number of LSTM Cells	Average Overall MAE	T_A Average MAE	T_B Average MAE
1	0.0017	0.0015	0.0018
2	0.0017	0.0015	0.0018
3	0.0027	0.0026	0.0027
4	0.0015	0.0014	0.0015
5	0.0019	0.0018	0.0019

Increasing the number of BiLSTM layers did not yield meaningful improvements in prediction accuracy. While a marginal reduction in MAE was observed for the four-layer configuration, the improvement was negligible relative to the associated increase in model complexity and training time. Therefore, stacking additional BiLSTM layers was not considered beneficial for this application.

Hidden Units

The number of hidden units governs the size of the internal cell state in the BiLSTM architecture. Increasing the number of hidden units enables the model to capture more complex temporal relationships but also increases computational cost. The number of hidden units was swept from 16—below the reference value—up to 1024, doubling at each step, while all other parameters were held constant.

Table 5.4: Hyperparameter Sweep — Hidden Units

Number of Hidden Units	Average Overall MAE	T_A Average MAE	T_B Average MAE
16	0.0026	0.0023	0.0029
32	0.0017	0.0016	0.0017
64	0.0017	0.0015	0.0018
128	0.0015	0.0014	0.0016
256	0.0015	0.0013	0.0017
512	0.0019	0.0021	0.0016
1024	NaN	NaN	NaN

Tuning the number of hidden units produced only minor improvements relative to the reference model. Performance improved modestly between 32 and 256 hidden units, after which the MAE increased. The 1024 hidden unit configuration failed to converge, yielding invalid outputs. Overall, increasing the number of hidden units provided diminishing returns while substantially increasing model complexity and training time.

Batch Size

Batch size determines how frequently the model weights and biases are updated during forward and backward propagation. Smaller batch sizes lead to more frequent weight

updates, which can improve generalization at the cost of longer training times. Larger batch sizes converge more quickly but may reduce generalization performance. Batch size was swept from 8 to 512 while all other parameters were held constant.

Table 5.5: Hyperparameter Sweep — Batch Size

Batch Size	Average Overall MAE	T_A Average MAE	T_B Average MAE
8	0.0019	0.0018	0.0020
16	0.0017	0.0015	0.0018
32	0.0014	0.0013	0.0015
64	0.0018	0.0017	0.0018
128	0.0019	0.0019	0.0019
256	0.0024	0.0023	0.0024
512	0.0022	0.0021	0.0023

Batch size had minimal impact on model performance. The lowest MAE was achieved at a batch size of 32, though the improvement relative to neighboring configurations was small. Larger batch sizes slightly degraded performance but remained within a narrow error range. Overall, batch size exerted limited influence on prediction accuracy.

Segment Size

The segment size determines the length of the input sequence fed into the model at each step. During preprocessing, the data is divided into fixed-length sequences; larger segment sizes provide more temporal context per sample but reduce the total number of available training sequences. Conversely, smaller segment sizes increase the number of training samples but may limit the model’s temporal modeling capability.

Table 5.6: Hyperparameter Sweep — Segment Size

Segment Size	Average Overall MAE	T_A Average MAE	T_B Average MAE
250	0.0020	0.0020	0.0020
500	0.0021	0.0020	0.0021
1000	0.0017	0.0015	0.0018
2500	0.0017	0.0018	0.0015
5000	0.0014	0.0014	0.0013
10000	0.0015	0.0013	0.0016

Increasing segment size generally led to modest improvements in performance, with the lowest MAE observed at larger segment sizes. However, the gains were marginal and came at the cost of increased memory usage and longer training times. Smaller segment sizes performed slightly worse but remained within a narrow error range.

Overlap %

Prediction overlap was also evaluated to determine whether overlapping prediction windows improved performance.

Table 5.7: Hyperparameter Sweep — Overlap

Prediction Overlap	Average Overall MAE	T_A Average MAE	T_B Average MAE
0%	0.0023	0.0021	0.0025
5%	0.0017	0.0015	0.0018
10%	0.0016	0.0014	0.0018
15%	0.0016	0.0014	0.0018
20%	0.0016	0.0014	0.0018
25%	0.0016	0.0014	0.0018

Introducing prediction overlap improved performance relative to the non-overlapping baseline, with the majority of the gain occurring between 0% and 10%. Beyond 10%, performance gains saturated, indicating diminishing returns for larger overlap values.

Model Selection

Based on the hyperparameter sweeps, several representative model configurations were selected for comparison. The reference model is included to provide a direct comparison between performance at the start and end of the tuning process. The model labeled “Best” was assembled by selecting, for each hyperparameter, the value that produced the lowest overall MAE. The model labeled “Worst” was assembled analogously using the values that produced the highest MAE. The “Middle” model was formed by selecting the values yielding MAE between the best and worst. Finally, the “Observation” model was assembled based on trends identified through inspection of the sweep results.

Table 5.8: Models Selected Hyper-Parameters and Results

Model	BiLSTM Cells	Hidden Units	Batch Size	Segment Size	Overlap	Average Overall MAE	T_A Average MAE	T_B Average MAE	Epochs
ref	1	64	16	1000	5%	0.0017	0.0015	0.0018	23
1 - Best	4	128	32	5000	10%	0.0017	0.0015	0.0018	65
2 - Worst	3	16	256	500	0%	0.0026	0.0025	0.0027	100
3 - Middle	2	32	128	2500	5%	0.0017	0.0016	0.0017	100
4 - Observation	1	128	32	5000	10%	0.0014	0.0012	0.0016	100

Overall, hyperparameter tuning demonstrated that model performance was relatively insensitive to architectural and training variations. Most configurations produced similar MAE values, indicating that the BiLSTM architecture is robust for this signal separation task when provided with two full-duplex tap signals as inputs. Consequently, simpler configurations were preferred when performance differences were negligible, reducing training time and computational complexity while maintaining comparable accuracy.

Changing Tap Points Comparison

While tap points A and B were the initial design choice due to their symmetrical placement relative to their respective endpoints, tap point D was also available for use. This section investigates whether the choice of tap point combination affects the prediction accuracy of the component signals.

Model 4, the “Observation” model, was identified as providing the best component signal reconstruction and was used as the baseline for comparison. Two additional models were trained using alternative tap point combinations: A and D , and B and D .

In addition to varying the tap points, the LSTM and GRU architectures were re-evaluated using the same hyperparameters as Model 4 to determine whether the superior signal replication of the Bi-LSTM model was attributable to more accurate decoding.

Table 5.9: Double Tap Point and RNN Comparison Models

Model	Average Overall MAE	T_A Average MAE	T_B Average MAE
Observation	0.0017	0.0015	0.0018
AD	0.0025	0.0022	0.0027
BD	0.0026	0.0025	0.0026
LSTM	0.0111	0.0101	0.0120
GRU	0.0107	0.0099	0.0115

Overall, the models using alternative tap point combinations performed reasonably well relative to one another and to the reference model. The predicted component signals remained accurate; however, a noticeable increase in MAE was observed compared to using tap points A and B . Additionally, these models required longer training times to converge.

This reduction in accuracy is likely attributable to the relative spacing of the tap points along the wire. Tap point D is centrally located, whereas A and B are positioned between D and the nearest endpoints. This difference alters the signal propagation relationships between the tapped signals and the component signals T_A and T_B , requiring the model to learn a more complex mapping to accurately reconstruct the component signals.

The LSTM and GRU models showed improved performance compared to their earlier reference configurations; however, their MAE remained significantly higher than that of the BiLSTM models.

Decoding Results

The ground-truth decoding results yielded a 93% decode accuracy on successfully identified frames. The MAE was used solely to characterize how closely the predicted

component signals matched the ground truth. The true measure of model performance was how well each configuration performed when subjected to the same decoding pipeline. Ideally, the predicted component signals would achieve decode accuracy at least comparable to that obtained on the ground-truth signals.

The decode results for the reference, best, worst, middle, observation, double tap comparison, and RNN comparison models are presented below. The same evaluation procedure applied to the ground-truth data was used here, and the same sources of error—PAM3 errors, SSD/ESD errors, and no frame detected—apply.

Table 5.10: Passive Double Tap Decode Results

Model	Accuracy	VALID Frames	Total Frames	PAM3 Errors	SSD/ESD Errors	No Frames Found
Ref	0.9423	702	745	22	21	142
Best	0.9516	708	744	15	21	151
Worst	0.7883	592	751	135	24	152
Mid	0.9464	706	746	17	23	160
Observation	0.9570	712	744	11	21	165
AD	0.9491	709	747	14	24	162
BD	0.9518	711	747	12	24	162
LSTM	0.9704	720	742	2	20	150
GRU	0.9689	717	740	3	20	147

These decoding results lead to a notable conclusion: every model except the “Worst” configuration outperformed the ground-truth decoding baseline, with all exceeding 93% accuracy. This outcome is likely attributable to the model predictions potentially improving waveform fidelity by producing more consistent transitions and amplitude levels than the raw ground-truth signals.

The Observation model achieved the highest decode accuracy among all double tap BiLSTM configurations; however, the overall highest accuracy was obtained by the

LSTM and GRU models. These models exhibited significantly higher MAE values yet achieved superior decode accuracy relative to the BiLSTM models—a finding that clearly demonstrates MAE alone is not a sufficient indicator of passive tap prediction quality. A possible explanation for this improved decoding performance is that the LSTM and GRU models, while not as close to the exact waveform shape, may produce outputs that more consistently hover near expected voltage levels, whereas the BiLSTM more accurately replicates the detailed waveform of the ground-truth signals.

Overall, a double tap RNN model can effectively serve as a passive tap for Automotive Ethernet and, in most configurations, can improve upon signal fidelity as evidenced by the higher decode accuracy achieved relative to the ground-truth baseline. These results indicate that ML is a viable approach for developing a passive tap for full-duplex communication systems.

PASSIVE SINGLE TAP

Introduction

The passive tap investigation demonstrated that a BiLSTM model could accurately separate component signals from two full-duplex tap points with relatively low complexity. The hyper-parameter sweep revealed that model performance was largely insensitive to architectural and training variations, suggesting that the two-tap configuration provided sufficient information for the model to easily identify the separation pattern.

This chapter investigates whether component signal separation remains feasible using only a single tap point. Reducing the number of required tap points has practical implications, as it simplifies the hardware requirements of the passive tap. However, the problem fundamentally changes in complexity; the model must now infer both component signals from a single observation, rather than leveraging two spatially distinct measurements of the wire.

The same methodology established in the passive tap investigation is followed here. A reference model is first established using the optimal configuration identified previously. A sweep of hyper-parameters is then performed to determine whether model complexity has a greater impact on performance in this more constrained setting. Finally, representative model configurations are selected and evaluated on decode accuracy, with other single tap points, LSTM, and GRU architectures included for comparison.

Reference Model

The reference model for the single-tap investigation uses the same hyperparameter configuration as the optimal model identified in the passive double tap chapter. This provides a meaningful baseline and allows direct comparison between the two-tap and single-tap

configurations under identical training conditions. Tap point D was selected to train the reference model as it was in the direct middle of the endpoints in the experiment and was thought to have the most success in recognizing equal but opposite shifts in the predictions.

Table 6.1: Single Tap Reference Model Hyper-Parameters

Number of LSTM Cells	Number of Hidden Units	Batch Size	Segment Size	Overlap %
1	128	32	5000	10%

Determining RNN Architecture

As in the passive tap investigation, LSTM, GRU, and BiLSTM architectures were evaluated using the reference model hyper-parameters to determine which model type best suits the single-tap separation task.

Table 6.2: Single Tap Model Type Comparison

Model Type	Average Overall MAE	T_A MAE	T_B Average MAE
LSTM	0.0845	0.0962	0.0728
GRU	0.0797	0.0918	0.0675
BiLSTM	0.0270	0.0273	0.0267

The Bi-LSTM had far better MAE values than the LSTM and GRU architecture. This is likely because the single-tap configurations are a much more complex task that the LSTM and GRU cannot handle like the Bi-LSTM. Therefore, the Bi-LSTM will be selected once again as the reference model for hyperparameter tuning. The LSTM and GRU will be trained

with identical hyperparameters as the best performing Bi-LSTM model for comparison in the decoding test to determine if a similar result can be obtained to the double-tap configuration.

Tuning Hyper-Parameters

The reference model served as the baseline for hyper-parameter tuning. The same sweep methodology from Chapter 5 was applied, varying one parameter at a time while holding all others constant. Given that the single-tap task presents a more complex separation problem, the impact of model capacity on performance was expected to be more pronounced than in the two-tap case.

Architecture

One notable addition to the Bi-LSTM architecture with multiple Bi-LSTM layers, is that the hidden units for each layer followed a funnel pattern, where the first layer had the set number of hidden units, and each layer proceeding had half the amount of hidden units compared to the previous layer. This is a typical architecture for ANNs.

Table 6.3: Single Tap — Modifying Hyper-Parameters: Architecture

Number of Bi-LSTM Cells	Average Overall MAE	T_A Average MAE	T_B Average MAE
1	0.0270	0.0273	0.0267
2	0.0034	0.0032	0.0036
3	0.0025	0.0024	0.0025
4	0.0049	0.0047	0.0051
5	0.0025	0.0022	0.0027

Network depth had a far greater impact on single-tap performance than in the double-tap case. Adding a second BiLSTM layer reduced MAE from 0.0270 to 0.0034, and a third

layer improved this further to 0.0025. This contrasts sharply with the double-tap results, where additional layers provided negligible benefit, highlighting that the single-tap problem requires greater model capacity. Beyond three layers, performance became inconsistent, likely due to optimization difficulties in deeper networks. The three-layer configuration was selected as the best balance between performance and training stability.

Hidden Units

Table 6.4: Single Tap — Modifying Hyper-Parameters: Hidden Units

Number of Hidden Units	Average Overall MAE	T_A Average MAE	T_B Average MAE
32	0.0492	0.0492	0.0492
64	0.0279	0.0279	0.0278
128	0.0270	0.0273	0.0267
256	0.0167	0.0169	0.0164
512	0.0106	0.0108	0.0103

Hidden unit count had a strong, consistent effect on performance — MAE decreased from 0.0492 at 32 units to 0.0106 at 512 with no sign of saturation. This contrasts with the double-tap case and suggests the model has not yet reached its representational ceiling. The 1024-unit configuration could not be evaluated due to GPU memory constraints, and further exploration with higher-capacity hardware is warranted.

Batch Size

Table 6.5: Single Tap — Modifying Hyper-Parameters: Batch Size

Batch Size	Average Overall MAE	T_A Average MAE	T_B Average MAE
8	0.0084	0.0085	0.0083
16	0.0122	0.0121	0.0123
32	0.0270	0.0273	0.0267
64	0.0276	0.0278	0.0274
128	0.0479	0.0482	0.0476

Batch size had a more significant effect here than in the double-tap case, with MAE decreasing consistently as batch size decreased from 0.0479 at 128 down to 0.0084 at 8. More frequent weight updates appear particularly beneficial for learning this more complex single-tap mapping. The smallest feasible batch size should be preferred, subject to memory and training time constraints.

Segment Size

Table 6.6: Single Tap — Modifying Hyper-Parameters: Segment Size

Segment Size	Average Overall MAE	T_A Average MAE	T_B Average MAE
500	0.0123	0.0127	0.0118
1000	0.0104	0.0105	0.0103
2500	0.0115	0.0114	0.0117
5000	0.0270	0.0273	0.0267
10000	0.0307	0.0308	0.0305
20000	0.0490	0.0494	0.0485

Segment size showed that is was optimal at 1000 samples, contrasting with the double-tap case where larger segments performed better. Smaller segments increase the total number of training sequences available, which appears to benefit convergence on the harder single-tap problem. At 500 samples performance degraded slightly, suggesting the model requires a minimum amount of temporal context per sequence. Beyond 1000, MAE increased steadily, reaching 0.0490 at 20000 samples.

Prediction Overlap

Table 6.7: Single Tap — Optimizing Overlap

Prediction Overlap	Average Overall MAE	T_A Average MAE	T_B Average MAE
0%	0.0286	0.0289	0.0282
5%	0.0270	0.0273	0.0267
10%	0.0270	0.0273	0.0267
15%	0.0270	0.0273	0.0267
20%	0.0270	0.0273	0.0267
25%	0.0270	0.0273	0.0267

Results closely mirrored the double-tap case. Any overlap improved performance relative to the zero-overlap baseline, but results saturated immediately at 5% with no further improvement beyond that point. A 10% overlap was retained for consistency with the double-tap configuration.

Selected Model Configurations

Based on the hyperparameter sweeps, several representative model configurations were selected for comparison. The reference model is included to provide a direct comparison between performance at the start and end of the tuning process. The model labeled “Best” was assembled by selecting, for each hyperparameter, the value that produced the lowest overall MAE. The model labeled “Worst” was assembled analogously using the values that produced the highest MAE. The “Middle” model was formed by selecting the values yielding MAE between the best and worst. Finally, the “Observation” model was assembled based on trends identified through inspection of the sweep results.

Table 6.8: Single Tap — Selected Models Hyper-Parameters and Results

Model	BiLSTM Cells	Hidden Units	Batch Size	Segment Size	Overlap	Average Overall MAE	T_A Average MAE	T_B Average MAE	Epochs
ref	1	128	32	5000	10%	0.0270	0.0273	0.0267	100
1 - Best	3	512	16	1000	10%	0.0053	0.0049	0.0057	10
2 - Worst	1	32	128	20000	0%	0.1506	0.1515	0.1497	100
3 - Mid	2	256	16	5000	10%	0.0022	0.0021	0.0022	100
4 - Observation	3	128	16	1000	10%	0.0032	0.0031	0.0033	10

The Middle model achieved the lowest overall MAE of 0.0022, with nearly identical performance across both component signals. Notably, this is comparable to the best double-tap configurations, suggesting that with sufficient model capacity, a single tap point can approach two-tap reconstruction quality. The Best model converged in only 10 epochs, suggesting early stopping terminated training prematurely despite its individually optimal hyperparameters. The Worst model confirmed that a shallow architecture combined with large batch size and segment size severely degrades performance, with MAE reaching 0.1506.

Comparison Models

Model 3 was selected to train the remaining models, using the same hyperparameters for training with tap point A , tap point B , and LSTM with tap point D , and a GRU with tap point D .

Table 6.9: Single Tap Point and RNN Comparison Models

Model	Average Overall MAE	T_A Average MAE	T_B Average MAE
Middle	0.0018	0.0018	0.0018
A	0.0017	0.0016	0.0018
B	0.0018	0.0016	0.0019
LSTM	0.0509	0.0553	0.0465
GRU	0.0838	0.0895	0.0780

The BiLSTM models using tap points D, A, and B achieved nearly identical MAE, all within 0.0001 of one another. This consistency across all three tap point locations suggests that for the single-tap BiLSTM configuration, the choice of tap point has minimal impact on signal reconstruction quality, in contrast to the double-tap case where tap point proximity to the endpoints influenced performance. The LSTM and GRU models, however, failed to learn the single-tap separation task effectively, producing MAE values of 0.0509 and 0.0838 respectively, more than an order of magnitude higher than the BiLSTM configurations. This stands in sharp contrast to the double-tap results, where LSTM and GRU achieved competitive MAE and even superior decode accuracy. The single-tap problem is fundamentally more complex, and the bidirectional processing capability of the BiLSTM appears essential for inferring both component signals from a single observation.

Decoding Results

Table 6.10: Single Tap Decode Results

Model	Accuracy	VALID Frames	Total Frames	PAM3 Errors	SSD/ESD Errors	No Frames Found
Ref	NaN	NaN	NaN	NaN	NaN	NaN
Best	12.1%	52	430	254	124	517
Worst	NaN	NaN	NaN	NaN	NaN	NaN
Mid	82.1%	527	642	56	59	164
Observation	54.6%	339	621	208	74	351
A	93.1%	648	696	18	30	181
B	96.0%	716	746	5	25	152
LSTM	10.2%	18	176	95	63	200
GRU	NaN	NaN	NaN	NaN	NaN	NaN

The single-tap decode results reveal a wide performance range across model configurations, and several results warrant careful interpretation. The Ref and Worst models produced NaN results, indicating complete failure to generate decodable signals, consistent with the expectation that poorly configured single-tap models lack the capacity to learn the separation task. The Best model, despite assembling individually optimal hyperparameters, achieved only 12.1% decode accuracy with a high PAM3 error count, reinforcing the earlier observation that early stopping terminated training prematurely at 10 epochs before the model fully converged.

The Middle model achieved 82.1% decode accuracy, a substantial result given the difficulty of the single-tap problem, though it falls short of the 93.0% ground-truth baseline. The Observation model reached only 54.6%, despite comparable MAE to the Middle model, again demonstrating that MAE is not a reliable predictor of decode performance.

The most striking results come from tap points A and B, which achieved decode accuracies of 93.1% and 96.0% respectively — matching and exceeding the ground-truth baseline. This is a remarkable finding: a single passive probe at either endpoint-proximal

tap point is sufficient to achieve decode accuracy comparable to the double-tap approach. The superior performance of tap points A and B over tap point D likely reflects their closer proximity to the individual endpoints, providing a signal with a more dominant contribution from one component, which reduces the complexity of the separation task for the model.

The LSTM model achieved only 10.2% accuracy and the GRU result was invalid, consistent with the MAE comparison showing these architectures are inadequate for single-tap separation. Overall, the single-tap results demonstrate that while the task is significantly more sensitive to configuration than the double-tap case, well-designed BiLSTM models with appropriate tap point selection can achieve ground-truth-level decode accuracy from a single passive observation point.

CONCLUSION

This thesis investigated the feasibility of using RNNs as a passive monitoring tool for full-duplex Automotive Ethernet communication. The core challenge addressed was the separation and decoding of superimposed component signals from a passively observed full-duplex PAM3 channel, without participation in link negotiation or knowledge of the transmitted data at either endpoint.

To support this investigation, a custom decoding pipeline was developed and validated against ground-truth data, achieving an overall frame decode accuracy of 93.0%. This pipeline handled all stages of the 100BASE-T1 receive chain, including clock recovery, PAM3 symbol extraction, frame boundary detection, LFSR-based descrambling, and CRC validation. The pipeline serves as the primary evaluation framework for all subsequent ML experiments.

A double-tap passive monitoring approach was then developed and evaluated. Bi-LSTM, LSTM, and GRU architectures were trained to predict the individual component signal T_A and T_B from two passively observed full-duplex signals. Hyperparameter tuning across architecture depth, hidden units, batch size, segment size, and prediction overlap demonstrated that model performance was relatively insensitive to architectural variation, with the Bi-LSTM consistently achieving the lowest MAE. However, a key finding emerged from the decoding evaluation. LSTM and GRU models, despite higher MAE relative to the Bi-LSTM, achieved superior frame decode accuracy of 97.0% and 96.9% respectively, exceeding the ground-truth baseline of 93.0%. The result demonstrated that MAE alone is not a sufficient indicator of passive tap quality, and that decode accuracy is the more meaningful measure for model performance.

A single-tap passive monitoring approach was then explored to define the limitations of ML based signal separation. The Bi-LSTM architecture was necessary for any tap point to

be successful given the more complex task. A respectable performance was achieved for all three single-tap points, with tap point B performing equally as well as the best double-tap configurations. These results were not replicated with tap points A or D even with the same model parameters, likely due to the stopping criteria limiting these models. Even though these models performed worse, as a whole, all single-tap decoding results provide a proof of concept that with a complex model architecture, signal separation is in fact possible with one tap point.

Research Questions Revisited

The research questions posed in Chapter 1 can now be addressed.

Can machine learning models predict component signals within a full-duplex Automotive Ethernet channel using only passively observed signals? Yes. Many RNN models successfully reconstructed component signals from passively observed full-duplex measurements, with the best configurations exceeding ground-truth decode accuracy.

Which RNN architecture provides the most effective performance? For signal replication as measured by MAE, the Bi-LSTM consistently outperformed the LSTM and GRU. However, for the more meaningful metric of frame decode accuracy, the LSTM and GRU achieved superior results, suggesting that exact waveform fidelity is less important than producing consistent, decodable signal levels.

Are the predicted component signals sufficiently accurate to allow successful decoding of Ethernet frames? Yes. The majority of double tap model configurations exceeded the 93.0% ground-truth decode accuracy baseline, confirming that machine learning predictions contain sufficient information to reconstruct valid Ethernet frames.

What limitations exist when applying machine learning to passive monitoring of full-duplex communication channels? The primary limitations identified were that for a double tap model, any configuration with overlapping predictions would result in

successful component signal prediction and even improve the decode accuracy. Single tap is even possible with the right model architecture to capture the more complex task. All single tap points produced respectable decode results, with just a much higher variation due to model training differences.

Future Work

The next step in this research project will be to apply what was learned from the ML based passive tap of Automotive Ethernet and apply it to Gigabit Ethernet (1000BASE-T). Gigabit Ethernet has full-duplex communication across four twisted-pair cables using PAM5 encoding. The increased signal complexity and the need to separate signals across multiple pairs represents a more challenging problem for developing a passive tap. The methodology developed for Automotive Ethernet should provide a strong foundation for solving the same problem for Gigabit Ethernet.

INL is working on providing datasets with more variable frames so that the decoding pipeline can be tested on a more practical networking environment. Evaluating performance across a more diverse dataset would further validate the developed decoding pipeline.

Finally, if Gigabit Ethernet ML passive decoding can be achieved, implementing a design on hardware and applying to an actual network environment would be achieving the ultimate goal of the project.

REFERENCES CITED

- [1] Ieee standard for ethernet. *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pages 1–7025, 2022. doi: 10.1109/IEEESTD.2022.9844436.
- [2] Ijaz Ahmad, Shariar Shahabuddin, Hassan Malik, Erkki Harjula, Teemu Leppänen, Lauri Lovén, Antti Anttonen, Ali Hassan Sodhro, Muhammad Mahtab Alam, Markku Juntti, Antti Ylä-Jääski, Thilo Sauter, Andrei Gurtov, Mika Ylianttila, and Jukka Riekk. Machine learning meets communication networks: Current trends and future challenges. *IEEE Access*, 8:223418–223460, 2020. doi: 10.1109/ACCESS.2020.3041765.
- [3] Mohammed M Alani. Guide to osi and tcp/ip models. 2014.
- [4] Gigabit Ethernet Alliance. Gigabit ethernet. 1997.
- [5] George W Benthien. Digital encoding and decoding, 2007.
- [6] Thomas Bressoud and David White. *The HyperText Transfer Protocol*, pages 609–648. Springer International Publishing, Cham, 2020. ISBN 978-3-030-54371-6. doi: 10.1007/978-3-030-54371-6_20. URL https://doi.org/10.1007/978-3-030-54371-6_20.
- [7] Broadcom Corporation. Open alliance broadr-reach® (oabr) physical layer transceiver specification for automotive applications, v3.2. Technical Specification V3.2, OPEN Alliance, June 2014. URL https://www.ieee802.org/3/1TPCESG/public/BroadR_Reach_Automotive_Spec_V3.2.pdf. Available online.
- [8] Yanpei Chen, Tracy Xiaoxiao Wang, and Randy H. Katz. Energy efficient ethernet encodings. In *2008 33rd IEEE Conference on Local Computer Networks (LCN)*, pages 122–129, 2008. doi: 10.1109/LCN.2008.4664160.
- [9] Bruno Dutertre. Formal modeling and analysis of the modbus protocol. In *International Conference on Critical Infrastructure Protection*, pages 189–204. Springer, 2007.
- [10] J. Everitt, J. Parker, P. Hurst, D. Nack, K. Konda, and C. Raad. A 10/100 mb/s cmos ethernet transceiver for 10baset, 100basex, and 100basefx. In *1998 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, ISSCC. First Edition (Cat. No.98CH36156)*, pages 210–211, 1998. doi: 10.1109/ISSCC.1998.672439.
- [11] Keith Michael Filler. *MACHINE LEARNING METHODS FOR DIGITAL SIGNAL SEPARATION*. PhD thesis, MONTANA STATE UNIVERSITY Bozeman, 2025.
- [12] Haoran Guang, Yongling He, Weifeng Gong, Bin Ma, Bowen Zheng, Yi Shi, Yaoguang Cao, and Shichun Yang. An intrusion detection method for automotive ethernet networks based on time series prediction and rule setting. In *2024 IEEE Smart World Congress (SWC)*, pages 1820–1827, 2024. doi: 10.1109/SWC62898.2024.00280.
- [13] Peter Hank, Steffen Müller, Ovidiu Vermesan, and Jeroen Van Den Keybus. Automotive ethernet: In-vehicle networking and smart mobility. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1735–1739, 2013. doi: 10.7873/DATE.2013.349.

- [14] M. Hatamian, O.E. Agazzi, J. Creigh, H. Samueli, A.J. Castellano, D. Kruse, A. Madiseti, N. Yousefi, K. Bult, P. Pai, M. Wakayama, M.M. McConnell, and M. Colombatto. Design considerations for gigabit ethernet 1000base-t twisted pair transceivers. In *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No.98CH36143)*, pages 335–342, 1998. doi: 10.1109/CICC.1998.694994.
- [15] Peter Hoeven. A frequency domain adaptive equalizer for gigabit automotive ethernet.
- [16] Ming-Feng Hsu, Yen-Liang Chen, Kai-Yuan Jheng, and An-Yeu Wu. A shortened impulse response filter (sirf) scheme for cost-effective echo canceller design of 10gbase-t ethernet system. In *2006 IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 309–312, 2006. doi: 10.1109/SIPS.2006.352600.
- [17] Jingyu Huang and R.R. Spencer. The design of analog front ends for 1000base-t receivers. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 50(10):675–684, 2003. doi: 10.1109/TCSII.2003.818392.
- [18] S. Huss, M. Mullen, C.T. Gray, R. Smith, M. Summers, J. Shafer, P. Heron, T. Sawinska, and J. Medero. A dsp based 10baset/100basetx ethernet transceiver in a 1.8 v, 0.18 μ m cmos technology. In *Proceedings of the IEEE 2001 Custom Integrated Circuits Conference (Cat. No.01CH37169)*, pages 135–138, 2001. doi: 10.1109/CICC.2001.929741.
- [19] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [20] Jia-Le Kang, Ming-Hui Zhang, Xiao-Peng Liu, and Chen He. Quantum secured 10-gbit/s ethernet passive optical access network and its machine-learning-assisted implementation. *Phys. Rev. A*, 109:022617, Feb 2024. doi: 10.1103/PhysRevA.109.022617. URL <https://link.aps.org/doi/10.1103/PhysRevA.109.022617>.
- [21] Dongyeong Kim and Dongweon Yoon. Blind estimation of self-synchronous scrambler in dsss systems. *IEEE Access*, 9:76976–76982, 2021. doi: 10.1109/ACCESS.2021.3083071.
- [22] Sumit Kumar, Sumit Dalal, and Vivek Dixit. The osi model: overview on the seven layers of computer networks. *International Journal of Computer Science and Information Technology Research*, 2(3):461–466, 2014.
- [23] Quoc V Le et al. A tutorial on deep learning part 1: Nonlinear classifiers and the backpropagation algorithm. *Mountain View, CA*, 2015.
- [24] Edward A Lee and David G Messerschmitt. Timing recovery. In *Digital Communication*, pages 560–583. Springer, 1988.
- [25] Sangkeum Lee, Suil Kim, Minah Seo, and Dongsoo Har. Synchronization of frequency hopping by lstm network for satellite communication system. *IEEE Communications Letters*, 23(11):2054–2058, 2019. doi: 10.1109/LCOMM.2019.2936019.

- [26] Shenghong Li and Ross D. Murch. Full-duplex wireless communication using transmitter output based echo cancellation. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, pages 1–5, 2011. doi: 10.1109/GLOCOM.2011.6133971.
- [27] Tu Li, Liu Jun'an, and Song Juanjuan. 16 times over-sampling technique based on the adc module of stellaristm family of chips. In Dehuai Zeng, editor, *Future Intelligent Information Systems*, pages 403–408, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19706-2.
- [28] Chi-Shiung Lin, Yu-Chun Lin, Shyh-Jye Jou, and Mun-Tian Shiou. Concurrent digital adaptive decision feedback equalizer for 10gbase-lx4 ethernet system. In *2007 IEEE Custom Integrated Circuits Conference*, pages 289–292, 2007. doi: 10.1109/CICC.2007.4405735.
- [29] Hsiu-Ping Lin, N.F. Chen, Jyh-Ting Lai, and An-Yeu Wu. 1000base-t gigabit ethernet baseband dsp ic design. In *2004 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages IV–401, 2004. doi: 10.1109/ISCAS.2004.1329025.
- [30] K. Mueller and M. Muller. Timing recovery in digital synchronous data receivers. *IEEE Transactions on Communications*, 24(5):516–531, 1976. doi: 10.1109/TCOM.1976.1093326.
- [31] Samreen Naeem, Aqib Ali, Sania Anam, and Muhammad Munawar Ahmed. An unsupervised machine learning algorithms: Comprehensive review. *International Journal of Computing and Digital Systems*, 2023.
- [32] Seol-Hyun Noh. Analysis of gradient vanishing of rnns and performance comparison. *Information*, 12(11):442, 2021.
- [33] Apena Waliu Olalekan, Olasunkanmi Omowumi Grace, and Salako Anuoluwapo. Performance evaluation of ethernet transmission using m-ary pulse amplitude modulation techniques. *Journal of Engineering Studies and Research*, 26(4):74–85, 2020.
- [34] Folorunso Y Osisanwo, Joseph ET Akinsola, Oludele Awodele, John O Hinmikaiye, Oluwole Olakanmi, Joseph Akinjobi, et al. Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138, 2017.
- [35] Timothy O'Shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017. doi: 10.1109/TCCN.2017.2758370.
- [36] Hui Pan, Junhua Tan, Evelyn Wang, Jingguang Wang, Karthik Swaminathan, Ramalingam Pandarinathan, Ramesh Pasagadugula, VamshiKrishna Yakkala, Mostafa Hammad, Karim Abdelhalim, Kaijun Li, Su Cui, Jing Wang, Ahmad Chini, Mehmet Tazebay, Suresh Venkatesan, Derek Tam, Ichiro Fujimori, and Kambiz Vakilian. 10.3

- an analog front-end for 100base-t1 automotive ethernet in 28nm cmos. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 186–187, 2016. doi: 10.1109/ISSCC.2016.7417969.
- [37] M. Petrovic and M. Aboelaze. Performance of tcp/udp under ad hoc ieee802.11. In *10th International Conference on Telecommunications, 2003. ICT 2003.*, volume 1, pages 700–708 vol.1, 2003. doi: 10.1109/ICTEL.2003.1191496.
- [38] Donovan Porter. 100base-t1 ethernet: the evolution of automotive networking. *Texas Instruments, Techn. Ber*, 2, 2018.
- [39] Andrew Pulver and Siwei Lyu. Lstm with working memory. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 845–851, 2017. doi: 10.1109/IJCNN.2017.7965940.
- [40] Andrew Pulver and Siwei Lyu. Lstm with working memory. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 845–851, 2017. doi: 10.1109/IJCNN.2017.7965940.
- [41] YCAP Reddy, P Viswanath, and B Eswara Reddy. Semi-supervised learning: A brief review. *Int. J. Eng. Technol*, 7(1.8):81, 2018.
- [42] Wojciech Samek, Slawomir Stanczak, and Thomas Wiegand. The convergence of machine learning and communications, 2017. URL <https://arxiv.org/abs/1708.08299>.
- [43] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.
- [44] Stephan Schreiner, Gunnar Armbrecht, Stephan Kunz, Thomas Schmid, and Michael Wollitzer. A broadband directional coupler for the signal integrity analysis on automotive high speed differential data links. In *2015 IEEE International Symposium on Electromagnetic Compatibility (EMC)*, pages 650–655, 2015. doi: 10.1109/ISEMC.2015.7256240.
- [45] Nirmal V. Shende, Özgür Gürbüz, and Elza Erkip. Half-duplex or full-duplex communications: Degrees of freedom analysis under self-interference. *IEEE Transactions on Wireless Communications*, 17(2):1081–1093, 2018. doi: 10.1109/TWC.2017.2774823.
- [46] Osvaldo Simeone. A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, 4(4):648–664, 2018. doi: 10.1109/TCCN.2018.2881442.
- [47] George Thomas. Introduction to the modbus protocol. *The Extension*, 9(4):1–4, 2008.
- [48] Jos Van Der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate. *arXiv preprint arXiv:1804.04849*, 2018.

- [49] Aida Varzaghani, Athos Kasapi, Dimitri N. Loizos, Song-Hee Paik, Shwetabh Verma, Sotirios Zogopoulos, and Stefanos Sidiropoulos. A 10.3-gs/s, 6-bit flash adc for 10g ethernet applications. *IEEE Journal of Solid-State Circuits*, 48(12):3038–3048, 2013. doi: 10.1109/JSSC.2013.2279419.
- [50] Shengjie Wang, Tianyi Zhou, and Jeff Bilmes. Bias also matters: Bias attribution for deep neural network explanation. In *International Conference on Machine Learning*, pages 6659–6667. PMLR, 2019.
- [51] Zhaoqiang Xia. *An Overview of Deep Learning*, pages 1–18. Springer Singapore, Singapore, 2019. ISBN 978-981-10-5152-4. doi: 10.1007/978-981-10-5152-4_1. URL https://doi.org/10.1007/978-981-10-5152-4_1.
- [52] Da Xu, Yuting Ye, and Chuanwei Ruan. Understanding the role of importance weighting for deep learning. *arXiv preprint arXiv:2103.15209*, 2021.
- [53] Mo Yang and Jing Wang. Adaptability of financial time series prediction based on bilstm. *Procedia Computer Science*, 199:18–25, 2022.
- [54] S Zargar. Introduction to sequence learning models: Rnn, lstm, gru. *Department of Mechanical and Aerospace Engineering, North Carolina State University*, 37988518, 2021.
- [55] Li Zhang, Wei Zhou, Jian Gang Guo, Yu Chen, Zhen Fang, Shi Quan Liu, Dong Wang, and Hua Xing Yang. A data and clock recovery circuit for gigabit ethernet. In *2025 IEEE 14th International Conference on Communications, Circuits and Systems (ICCCAS)*, pages 226–230, 2025. doi: 10.1109/ICCCAS65806.2025.11102357.
- [56] W.-P. Zhu, M.O. Ahmad, and M.N.S. Swamy. A fully digital timing recovery scheme using two samples per symbol. In *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, volume 2, pages 421–424 vol. 2, 2001. doi: 10.1109/ISCAS.2001.921097.