ABSTRACT

LI, BOWEN. High-speed Receiver Behavioral Modeling Using Machine Learning. (Under the direction of Dr. Paul D. Franzon).

In the high-speed Serializer/Deserializer (SerDes) application, with the rising frequencies, signal integrity analysis becomes challenging. To simulate the performance of the SerDes link, behaviors of the transceiver need to be modeled accurately. In this research, we focus on the high-speed receiver behavioral modeling.

Nowadays, the IBIS-AMI model becomes the most efficient simulation model for the high-speed SerDes simulation. The IBIS-AMI model can finish one simulation in about 30 minutes for each channel. After the simulation, users can get the transient simulation results, namely transient waveform, eye diagram and bathtub curve, and adaptation simulation results, such as CTLE and DFE settings at the receiver.

However, the IBIS-AMI model can only do adaptation and transient simulation at the same time, which would be inefficient in two scenarios. The first scenario is that users only want to get transient simulation results under one receiver setting to see the eye-opening or bathtub curve information. The second scenario is that users only want to know the receiver adaptation codes, which can represent the equalization status of the SerDes link. In both scenarios, people have to wait for 30 minutes and get some redundant simulation results. In that case, individual behavioral models need to be built for these two receiver simulations, while the simulation speeds of the models should be faster than the IBIS-AMI model.

In this research, as for the transient behavioral modeling, we propose a method, called adaptive-ordered system identification model, to simulate the transient simulation results of the high-speed receiver. This modeling method would self-adapt the model complexity and has a fast

simulation speed. The proposed modeling method can provide high-precision transient waveform, eye diagram, and bathtub curve predictions.

As for the adaptation behavioral modeling, we propose a Cascaded Deep Learning (CDL) modeling mechanism to show a parallel approach to modeling adaptive SerDes behavior effectively. Specifically, the proposed modeling methodology uses a cascaded deep learning model structure to predict the receiver adaptation codes. The modeling method shows high correlations with the real adaptation results. Meanwhile, this modeling mechanism has good scalability, which can learn adaptation behaviors from various SerDes designs.

For the simulation speed, the proposed adaptive-ordered system identification model and CDL model are 180 times and 900 times faster than the state-of-the-art IBIS-AMI model.

High-speed Receiver Behavioral Modeling Using Machine Learning

by
Bowen Li

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina
2020

APPROVED BY:

Paul D. Franzon Committee Chair	Brian Floyd
Rhett Davis	Min Chi

DEDICATION

Live a life you will remember.

BIOGRAPHY

Bowen Li was born in December 1991, Shijiazhuang, Hebei, China. He received his bachelor's degree from Beijing University of Posts and Telecommunications in June 2014 and his master's degree from North Carolina State University in May 2018. In August 2015, he started his graduate research work with Dr. Paul Franzon in the area of machine learning in Electronic design automation (EDA) in North Carolina State University. During his Ph.D., he interned at Hewlett Packard Enterprise and Samsung. His research topic is high-speed receiver behavioral modeling using machine learning.

ACKNOWLEDGMENTS

I want to express my sincere gratitude to my research advisor, Dr. Paul Franzon, for his guideline, understanding and support. I also want to thank him for introducing me to the machine learning related research topic and providing me the opportunity to join the CAMEL center. Furthermore, I want to thank my committee member, Dr. Brian Floyd, Dr. Rhett Davis and Dr. Min Chi for insightful advice and understanding during the research work.

I'm also grateful to the following people and organizations, without whom I could not have made this accomplishment. These include Dr. Brandon Jiao, who inspired me in the SerDes design area and helped me during my research; Dr. Yongjin Choi, who was the mentor during my very first internship in my life; my group fellows Weiyi Qi, Sumon Dey, Weifu Li, Jong Beom Park, Zhao Wang, Bill Huggins, Yi Wang, and Luis Francisco, Yuejiang Wen, who have made special pieces of the beautiful memory of the past years; and my friends from other research groups Junyu Shen, Yuan Chang, Ruonan Yang, who support me during my Ph.D. research.

Finally, I would like to give my most sincere gratefulness to my parents. Their love and support have always been my primary motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1. Introduction

With the increasing data rate and distance of the wireline communication system, signal integrity analysis becomes challenging in high-speed Serializer-Deserializer (SerDes) links. The performance of the SerDes link needs to be analyzed and simulated efficiently. To analyze and simulate the performance of the entire SerDes link, behaviors of the transmitter (TX) and receiver (RX) need to be accurately modeled. In the SerDes link simulation, the most challenge part is to build a receiver behavioral model. Nowadays, because of the low target serial link error rate, for example, 1e-15, the existing simulation tools, like SPICE simulation, and traditional IBIS model, cannot provide fast and accurate simulations.

In that case, the IBIS Algorithmic Modeling Interface (IBIS-AMI) model is proposed [15] [33]. The IBIS-AMI model is a powerful method to incorporate SerDes and channel models into a unified simulation environment that protects vendors' intellectual property (IP) [19]. The IBIS-AMI is a behavior model, which is generated by the semiconductor vendors and for the customers to do the simulation. However, the IBIS-AMI model can only do adaptation and transient simulation at the same time, which would be inefficient in some situations. And the simulation speed of the IBIS-AMI model is 30 minutes for each run. Hence, we need to find a way to build behavioral models for these two simulations separately, while the performance of the behavioral models should be better than the IBIS-AMI model.

In this work, we focus on modeling the receiver behavior using machine learning methods, which can provide fast and high-precision simulation. Two behavioral models are built for these two types of simulations.

1.1. Motivation

In the transceiver or SerDes industry, there is one efficient way to do the transceiver simulation, which is the IBIS-AMI model. The IBIS-AMI model is a behavior model and shows a high correlation with the on-die circuit. The IBIS-AMI model can do signal integrity analysis in about 30 minutes for each channel. After the simulation, users can get the equalized transient simulation results, namely transient waveform, eye diagram and bathtub curve, and adaptation simulation results, such as adaptation codes at the receiver. However, the IBIS-AMI model can only do adaptation and transient simulation at the same time, which would be inefficient in two scenarios. The first scenario is that users want to get transient simulation results to see the eye-opening or bathtub curve information. The second scenario is that users only need to know the receiver adaptation codes, which represent the equalization status of the SerDes link. In that case, we need to find a way to build behavioral models for these two simulations separately, while the performance of the behavioral models should be the same or better than the IBIS-AMI model.

Recently, some works have been done in high-speed receiver behavioral modeling to speedup post-silicon validation. Those related work can be divided into two research areas. The first area is to predict eye-opening using machine learning models. Neural networks (NN) [12] or support vector machine (SVM) [28] [29] are trained to predict the eye height and eye width after the receiver equalization using the circuit features. However, those methods are grey-box modeling, which needs to know the inner structure of the circuit. In a SerDes system, there are approximately thousands of circuit features impacting the recovered signal in SerDes receiver. The published grey-box methods take a couple out of the thousands of features to build the ML model and do prediction. Second, recurrent neural networks (RNN) models [21] are used to predict the equalized

receiver output waveforms according to the receiver input waveforms. Those methods are black-box modeling, which doesn't need any internal information of the circuit.

For the published gray-box modeling, the tradeoff is the more features adopted in the ML model the more accurate the prediction but the model complexity and simulation time increase. The accuracy of the grey-box method will be sacrificed to balance the model complexity as well as simulation time. What's more, the gray-box modeling methods can only predict eye height and width, which is only a small part of the whole picture of the equalized signal inside the receiver. For the published black-box modeling, there are no modeling methods which can only train the model once and predict the transient waveforms, not from the same training pseudorandom binary sequence (PRBS) data and training channel. Moreover, some publications provide prediction accuracy while some do not. The provided accuracy numbers are all about eye height and eye width prediction without giving the accuracy number for predicted transient waveforms. Besides, no publications are found to do bathtubs prediction. The bathtub prediction is based on the eye diagram constructed from long enough transient waveforms. Generally, it needs about a million bits. The ML model complexity determines the model prediction/simulation time and consequently, the total length of the transient waveform prediction can be obtained with reasonable simulation time. All the latest published work can predict much fewer bits required for bathtub prediction. The proposed modeling method in this work can predict long bits in a short time and shows a high-precision prediction of the transient waveform, eye diagram, and bathtub. The proposed modeling method is a black-box modeling approach and provides a fast and high-precision, which can train the model once and predict transient waveforms, eye diagram and bathtub curve using different PRBS data patterns and different channels. The black-box method does not consider any SerDes circuit structure or parameter to build the model. As a result, the black-box method accuracy is not

limited by SerDes circuit details. To meet all the functions in the receiver, the machine-learning-based receiver model can provide transient waveform, eye diagram, and bathtub curve predictions at the same time.

The receiver adaptation process is another important feature in the high-speed transceiver simulation. In the transceiver or SerDes industry, there is only one way in the modeling method, called the IBIS-AMI model, to give the users the adaptation results, which is a part of the IBIS standard. However, the simulation speed of the adaptation process is very time-consuming. If the transmitter (TX) or the channel is changed, engineers have to wait for a half-hour to see the adaptation results using the IBIS-AMI model. Moreover, designing an IBIS-AMI model needs detailed information about the circuit design and lots of trials and errors to improve model performance. Currently, no research focuses on equalization adaptation modeling using machine learning technique.

In this research, machine learning (ML) models are used to predict equalized results, namely transient waveforms, eye diagrams, and bathtub curves, and the adaptation codes using the receiver input waveforms. The design and simulation process of the proposed modeling methods are much faster than the IBIS-AMI model.

1.2. Contribution

In this work, an adaptive-ordered system identification model is proposed for the receiver equalized result prediction. The main contributions of the proposed modeling method can be summarized as below:

1. Self-adaptive model complexity. The proposed modeling method can quickly predict long-bits transient waveforms for a high-speed receiver.

2. The proposed model can provide transient waveform, eye diagram, and bathtub curve simulation at the same time.

3. No re-train is needed to predict over various channels and no accuracy degradation when changing the data pattern, transmitter settings, and channels.

4. Fast training and prediction time compared to existing modeling methods. The simulation speed is much faster than the IBIS-AMI model.

The Cascaded Deep Learning (CDL) modeling mechanism is proposed to show a parallel approach to modeling adaptive SerDes behavior effectively. The main contributions of the CDL modeling method in this work can be summarized as below:

1. Originality. No previous work focuses on the SerDes receiver adaptation modeling using a machine learning approach.

2. The modeling mechanism can do cascaded prediction.

3. The proposed model provides high-precision receiver adaptation simulation results.

4. The simulation speed is much faster than the IBIS-AMI model.

5. Good scalability. This modeling mechanism can learn adaptation behaviors from different SerDes designs.

1.3. Organization

We organize the rest as follows. Chapter 2 introduce the fundamentals of the SerDes link, and the machine learning models we used. In Chapter 3, we obtain the related state-of-arts research on the high-speed receiver behavioral modeling and compare those work with our proposed method.

Chapter 4 proposes the adaptive-ordered system identification modeling method for the equalized receiver result prediction.

Chapter 5 presents the CDL modeling mechanism for receiver adaptation behavior. We will compare the results with the on-die circuit adaptation codes.

Chapter 6 concludes the work we've done for the high-speed receiver behavioral modeling and discusses the potential future work.

1.4. Abbreviations

SerDes	Serializer-Deserializer
TX	Transmitter
RX	Receiver
LTI	linear time-invariant
IBIS	Input/output Buffer Information Specification
IBIS-AMI	IBIS Algorithmic Modeling Interface
NN	Neural network
SVM	Support Vector Machine
ISI	Inter-symbol interference
CTLE	continuous time linear equalizer
DFE	decision feedback equalizer
FFE	feed-forward equalization
AGC	auto gain control
SNR	signal noise ratio
UI	Unit interval
PCB	Printed Circuit Board
ARX	AutoRegressive eXternal input
ARMAX	AutoRegressive Moving Average eXternal input
MLP	Multilayer perceptron
DNN	Deep neural networks

ReLU	Rectified linear unit
SGD	Stochastic gradient descent
LSTM	Long short-term memory
RNN	Recurrent neural networks
VRNN	Vanilla RNN
ANN	Artificial Neural Networks
BER	Bit error rate
LMS	least mean square
NRMSA	Normalized root mean square accuracy
100GE	100 Gigabit Ethernet
KH	High-frequency gain
KL	Low-frequency gain
VSR	Very short reach
SR	Short reach
MR	Medium reach
BERT	Bit error ratio testers
DUT	Device under test
PDF	Probability density function
DJ	Deterministic jitter
RJ	Random jitter
ML	Machine learning
NNARX	ARX model based on neural networks
ANNARX	Adaptive-ordered NNARX
PCC	Pearson Correlation Coefficient
BayesOpt	Bayesian Optimization

HDL	Hardware description language
CDL	Cascaded Deep Learning
IP	Intellectual property
PVT	Process, voltage, and temperature
PRBS	Pseudorandom binary sequence
SBR	Single bit response
LPR	Long pulse response
MSE	Mean square error
PTL	Predicted target library
CNN	convolutional neural networks
SD	Standard deviation

Chapter 2. Background and Fundamentals

In this chapter, we will describe the fundamentals of the SerDes link, IBIS-AMI model, and machine learning models used in this research.

2.1. SerDes Background

Signal integrity analysis becomes challenging in high-speed SerDes links. A basic SerDes link has a transmitter (TX), a channel, and a receiver (RX). The receiver consists of a continuous time linear equalizer (CTLE) and a decision feedback equalizer (DFE), which are used to mitigate inter-symbol interference (ISI). An ideal cable could propagate all frequency contents without any loss. TX feed-forward equalization (FFE) acts as a FIR filter and pre-distorts transmitted pulse in order to invert channel distortion. At receiver side, RX CTLE and DFE are implemented as part of receiver circuits and flatten the system response through conditioning the receiving signal. Figure 2.1 shows a high-speed electrical link using TX FFE equalization and RX CTLE+DFE equalization.

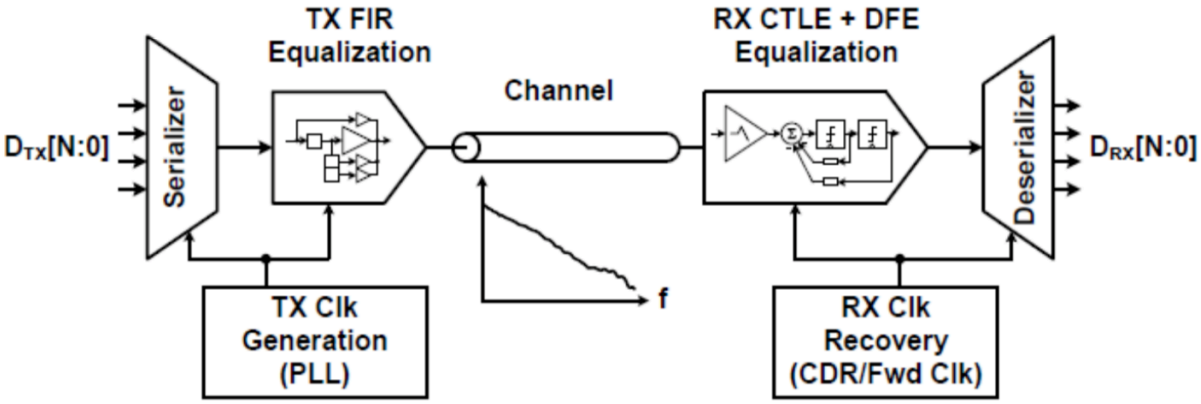


Figure 2.1. High-Speed Electrical Link with Equalization Schemes

We will briefly introduce the TX technology and talk about CTLE and DFE technology next. Transmit equalization is the most common technique in high-speed links design. It is usually

implemented through a FIR filter. It pre-distorts or shapes the data over several bit periods in order to invert the channel loss/distortion. The low frequency contents get de-emphasized in order to flatten the channel response. The advantages of the TX are: 1) High speed DAC is relatively easy to implement compared with receiver high speed ADC; 2) TX FFE can cancel pre-cursor ISI; 3) Due to the digital nature of the TX FFE, the noise is not amplified. The disadvantages are: 1) To flatten the channel response, low frequency content is attenuated due to the peak-power limitation. 2) To tune the FIR taps, a feedback path from receiver side is required to detect channel response.

Next, CTLE and DFE technology are introduced.

2.1.1. Basics of CTLE

CTLE, also known as linear equalizer, filters RX input signal by relatively attenuating the low frequency content and boosting high frequency content attenuated through the channel. It introduces zeros to offset the frequency-dependent channel loss. Generally, the CTLE is preceded and followed by an auto gain control (AGC) circuit to bring the signal to the appropriate level to achieve acceptable signal noise ratio (SNR) and to not amplify the signal too much to go too much beyond the nonlinear range. While boosting high frequency signal, CTLE could potentially amplify noise and crosstalk depending on the required CTLE settings. This calls for more precautions in using CTLE for certain applications, for example, when the environment is noisy. An example of CTLE effect in reducing channel ISI is illustrated in Figure 2.2 [24].

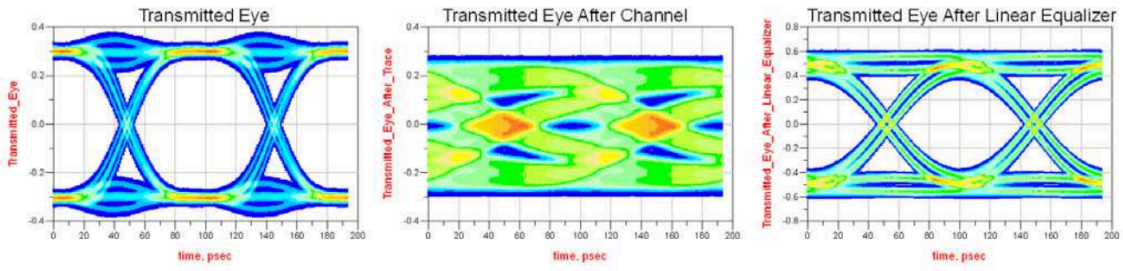
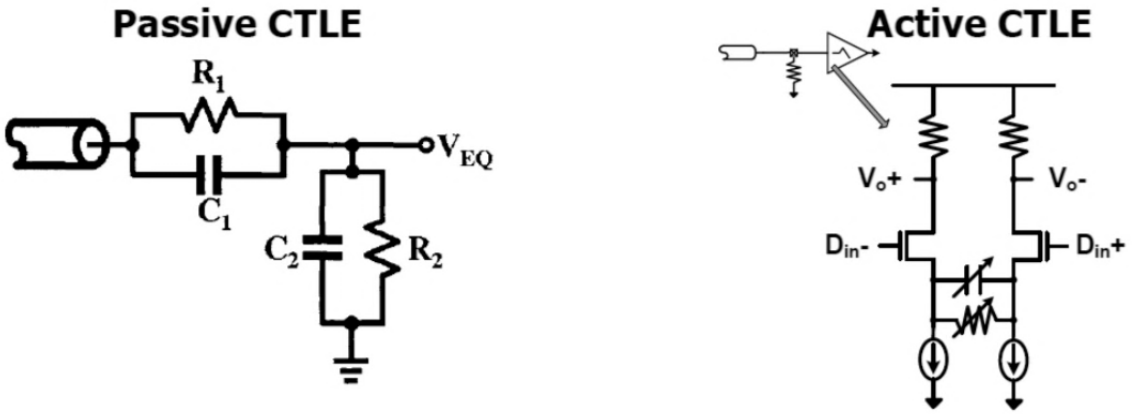


Figure 2.2. CTLE effect illustration

CTLE can be divided into two types, passive [22] and active filter [23]. Both passive and active filter can realize high-pass transfer function to compensate for the channel loss as shown in Figure 2.3. Both pre-cursor and long-tail ISI can be cancelled using the linear equalizer.



(a) Passive CTLE (b) Active CTLE

Figure 2.3. CTLE circuit structure

The passive CTLE is the combination of passive low pass filter and high pass filter. Active CTLE can be implemented through a differential pair with RC degeneration with gain at Nyquist frequency as shown in Figure 2.3 (b). At the high frequency, degeneration capacitor shorts the degeneration resistor and creates peaking. The peaking and DC gain can be tuned through adjustment of degeneration resistor and capacitor.

2.1.2. Basics of DFE

DFE is commonly implemented in high-speed links receiver side. DFE would subtract out channel impulse responses from previous data bits to zero out ISI impacts on the current bit. If the DFE has N taps, then the previous n-bits induced ISI can be mainly removed. The basic DFE working mechanism is shown in Figure 2.4 [25].

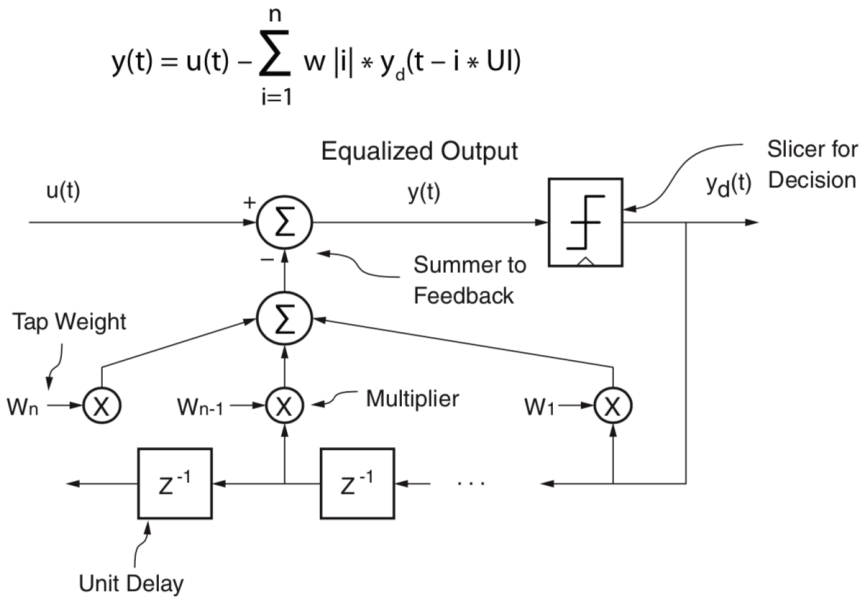


Figure 2.4. Basic DFE working mechanism

DFE can boost high frequency content without noise and crosstalk amplification. For example, Figure 2.5 shows a single bit response. The blue line is the original signal after the channel. Because the long tail would interfere upcoming signals, the signal after the main cursor should drop down quickly. DFE tap will erase the long tail by dragging post cursors down to zero. If we have 2 DFE taps in the receiver, we can erase 2 UIs tail. The distance a1 and a2, in Figure 2.5, directly influence the DFE tap1 and tap2 value. Typically, each DFE tap value has a very strong relation with the post cursor information. Figure 2.6 [25] shows two typical DFE corrected data eyes. On the left is for the case when the dominant post-cursor ISI is positive, while on the right negative. The left chart implies the congregated equalization before DFE is not enough, or under-equalized, while the right chart indicates the congregated equalization is excessive, or over-equalized.

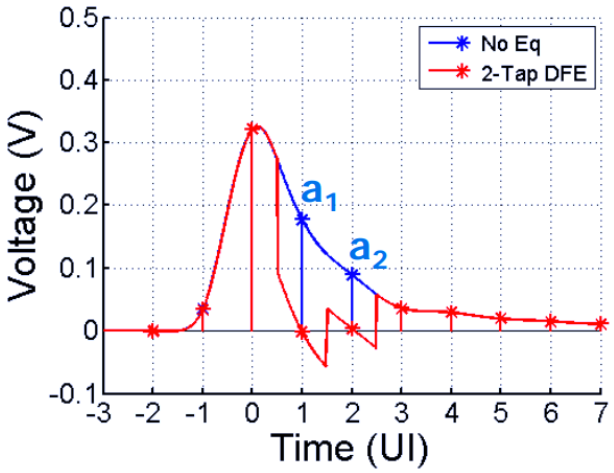


Figure 2.5. DFE tap function

Slicer makes a symbol decision without amplifying noise. The results are fed back to the slicer input through a FIR filter to cancel post-cursor ISI. The major challenge in DFE implementation is the closing timing on the first tap feedback, which must be done in one-bit period or unit interval (UI).

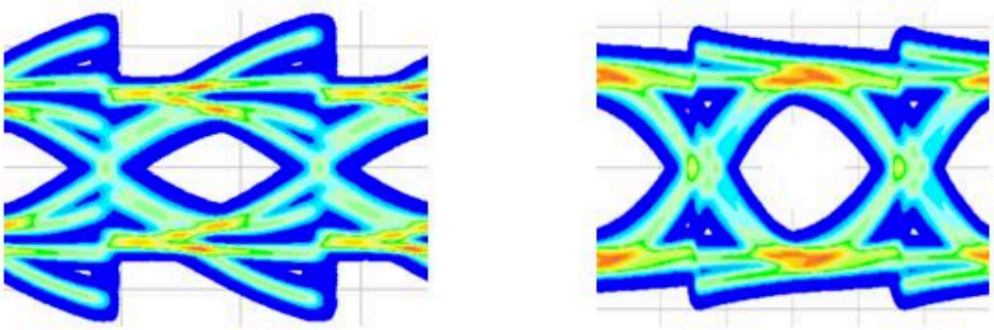


Figure 2.6. Typical eye diagrams after DFE

2.2. IBIS-AMI model introduction

To analyze and simulate the performance of the entire SerDes link, behaviors of the TX and RX need to be accurately modeled. Impairments should be considered inside the model. However, such information is typically proprietary to SerDes vendors and unavailable to the user of the model [19]. To deal with this problem, the IBIS Algorithmic Modeling Interface (IBIS-AMI)

model is proposed. The IBIS-AMI model is a powerful method to incorporate SerDes and channel models into a unified simulation environment which protects vendors' intellectual property. The IBIS-AMI model is a behavior model and shows a high correlation with the on-die circuit.

2.2.1. How does the IBIS-AMI model work

Input/output Buffer Information Specification (IBIS) models were first generated by Intel in 1993. As the chip designs became more and more complex, traditional IBIS models could no longer keep up with the DSP design. Vendors were again providing their own encrypted models which were platform specific. Thus, the IBIS community worked to extend the IBIS models and IBIS-AMI models were born. IBIS-AMI model represents an important milestone in the IBIS mixed-signal evolution [31].

AMI stands for Algorithmic Modeling Interface. It is designed to handle modeling of the algorithmic functions of an I/O. IBIS-AMI models provide the end user with the model portability that they need while ensuring the vendors that their IP is protected.

Figure 2.7 shows the same system with block level descriptions [34]. The channel contains many parts which may be individually modeled. This includes the connectors, Printed Circuit Board (PCB) traces, vias, and package models. The composite channel includes all of these with the addition of the analog IBIS buffer models. The IBIS-AMI model has two modes, namely statistical and time-domain simulation [16] [32]. Channel simulators make the assumption that this composite channel is linear time invariant (LTI) system. In both modes, the IBIS-AMI simulations begin by characterizing the channel's impulse response in the time domain. This is typically accomplished by generating a Heaviside step function at the transmitter's analog buffer and converting the response at the receiver's analog buffer by calculating the impulse response using the first-order derivative of the step response. With the impulse response of the analog channel, in

IBIS-AMI terminology, an IBIS-AMI simulation processes the effects of the models' filtering functions quite differently for time-domain or statistical methods.

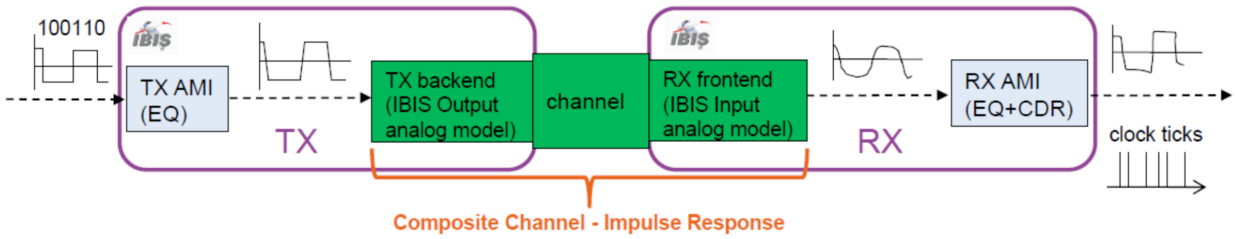


Figure 2.7. IBIS-AMI simulation description

2.2.2. IBIS-AMI model Development Challenges

The main challenge for the IBIS-AMI model development is that it requires lots of experienced experts and lots of trial and errors to develop a model [34]. High speed mixed signal IC designers and architects are not programmers. They are better equipped at dealing with various circuit design or algorithm developments. However, IBIS-AMI model development requires much different skills, such as skill in C/C++ coding, skill with the IBIS-AMI standard, skill on parsing IBIS files. Oftentimes, the circuit designers or architects become AMI model developers and are tasked to pull together disjoint pieces of their IC designs from incomplete IC specifications, from Spice simulations and from cryptic MATLAB/C++ code [30]. These disjoint pieces may lead to incomplete AMI models.

Moreover, designing an IBIS-AMI model needs detailed information about the circuit design and lots of trial & errors to improve model performance. Normally, it takes a long design cycle, for example, a half year, to build an IBIS-AMI model for one particular design.

2.3. System identification modeling approach

System identification is used to build models for dynamic systems based on measured input and output pairs. A system identification model is more like a map between a set of explanatory variables and a set of predicted variables [1] [2].

Generally, system identification models can be divided into two types, linear system identification modeling, and nonlinear system identification modeling.

2.3.1. Linear System Identification Model

If one system can be presented as the following form, this system is called linear system.

$$\mathrm{y}(t) = \mathrm{G}(q^{-1})\mathrm{u}(t) + \mathrm{H}(q^{-1})\mathrm{e}(t) \tag{2.1}$$

Where G and H are transfer functions in the time delay operator, q^{-1}. e(t) is the white noise signal, which is independent of model inputs and outputs. The delay operator is shown below:

$$q^{-d}x(t) = x(t-d) \tag{2.2}$$

Where d is how many sampling points delay between input and output. The system identification model is to use current and previous inputs and errors to predict the current output.

The basic configuration of linear system identification is shown in Figure 2.8. There are two main linear system identification structures, AutoRegressive eXternal input (ARX) model and AutoRegressive Moving Average eXternal input (ARMAX) model.

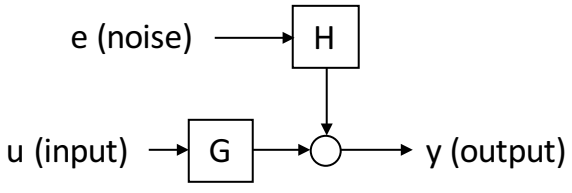


Figure 2.8. Linear system identification configuration

In ARX model, the current output is related to the previous inputs and outputs. It doesn't consider white-noise disturbance. The ARX model can be obtained as

$$A(q)y(t) = B(q)u(t - n_k) \tag{2.3}$$

where $A(q) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}$, $B(q) = b_1 + b_2 q^{-1} + \cdots + b_{n_b} q^{-n_b+1}$.

In ARMAX model, the current output is related to the previous inputs, the previous outputs and current and previous white-noise disturbance value. This means ARMAX model is more general than the ARX model. The ARMAX model structure can be represented as

$$A(q)y(t) = B(q)u(t - n_k) + C(q)e(t) \tag{2.4}$$

where $A(q) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}$, $B(q) = b_1 + b_2 q^{-1} + \cdots + b_{n_b} q^{-n_b+1}$, $C(q) = 1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}$.

2.3.2. Nonlinear system identification model

Nonlinear system identification model reuses the input structures from linear models and changes the internal structure to a feedforward multilayer perceptron (MLP) network [3]. Nonlinear system identification model is more general than linear system identification models. First, the internal structure is flexible to handle very complex nonlinear system. Second, it is easy to handle by end users. Third, it is suitable for the design of control systems. The nonlinear system identification formula is shown below:

$$y(t) = g[\varphi(t, \theta)] + e(t) \tag{2.5}$$

where $\varphi(t, \theta)$ is the regression vector and θ is the weights in the neural networks, and g is the feedforward neural networks.

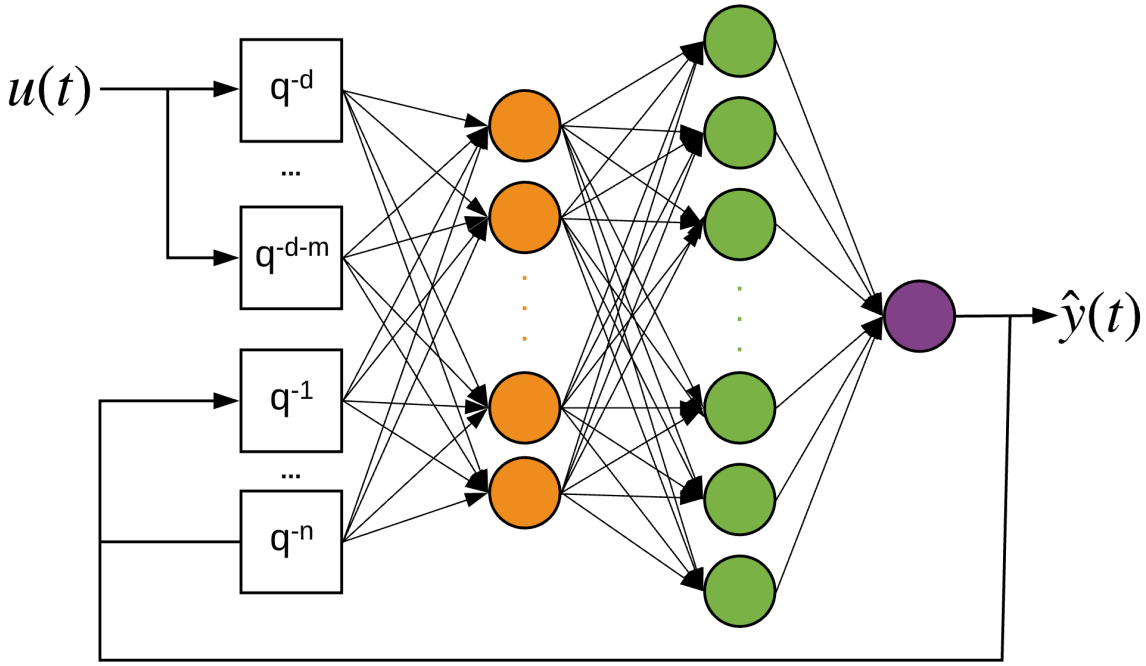


Figure 2.9. NNARX structure

The ARX model based on neural networks (NNARX) model structure is shown in Figure 2.9 [3]. The training process of the NNARX model is shown below:

1. The input and output orders are set; the weights and bias are initialed in the neural networks.
2. After initialization, the model will do forward propagation and calculate the final output.
3. After that, it will calculate the loss function. In this work, mean square error (MSE) is set as the loss function, which can be calculated as:

$$Loss = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2.6}$$

where y_i is the real output at sampling point i; $\hat{y}_i$ is the predicted value from the model at sampling point i. m is the total number of data points.

4. Next, the model will update the weights and bias in the neural networks by the backpropagation.

$$\widehat{W^l} = W^l - \alpha * dW^l \qquad (2.7)$$
$$\widehat{b^l} = b^l - \alpha * db^l$$

W^l and b^l are the current weights and bias at each layer l. dW^l and db^l are partial derivatives of loss function. α is the learning rate. $\widehat{W^l}$ and $\widehat{b^l}$ are updated weights and bias at each layer l.

5. After updating the weights and bias, it will go back to step 2 and repeat until the loss function is very small or the max number of learning step is reached.

This model structure is similar to the DFE structure in a receiver. The DFE has output feedbacks, which would influence the current output value. Meantime, the NNARX also has output feedbacks. The DFE has a slicer, which is a nonlinear component and used to quantize the inputs. In NNARX model, each neuron has an activation function, which is also nonlinear.

2.4. Deep learning model

2.4.1. Deep neural networks model

Neural networks model is a technology to mimic the activity of the human brain [26]. It can learn a black-box system. A simple application of the neural networks to do the data analysis is called MLP. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function, for example, sigmoid or tanh function. Each node in the hidden layer is a function of the nodes in the previous layer. The output node is a function of the nodes in the hidden layer. The number of neurons in the output layer depends on the number of outputs user set. MLP uses backpropagation for the training process. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Compared with the MLP, deep neural networks (DNN) [27] increases the hierarchy of complexity and abstraction. Figure 2.10 shows the DNN structure. Each layer applies a nonlinear transformation onto its input and creates a statistical model as output from what it learned. The input features are received by the input layer and passed into the first hidden layer. Each neuron in the hidden layers has weights and biases. Each neuron has an activation function which is used to standardize the output from the neuron. The "Deep" in deep neural networks means there is more than one hidden layer in DNN. The output layer returns the output data. Until the output has reached an acceptable level of accuracy, training epochs will be continued.

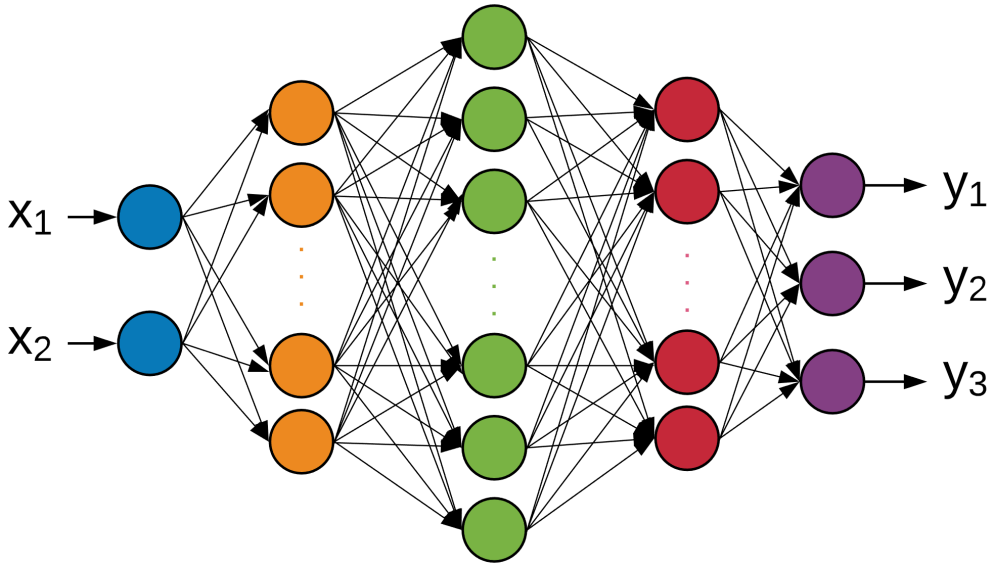


Figure 2.10. Deep Neural Networks structure

The number of neurons in each layer can be represented as $\{\mathrm{L}\} = (L^{in}, L^1, \ldots, L^h, L^{out})$, where L^{in}, L^h, L^{out} are the number of neurons in the input layer, h^{th} hidden layer, and output layer respectively. The input of the h^{th} hidden layer can be calculated by

$$\{z^h\} = \{x^{h-1}\}W^h \tag{2.8}$$

where W^h is a matrix which contains the weights between the output of the $(h-1)^{th}$ layer and h^{th} layer. The output vector $\{x^h\}$ of the h^{th} hidden layer is represented as

$$\{x^h\} = f_a(\{z^h\} + \{b^h\}) \tag{2.9}$$

where f_a is the activation function. In this work, rectified linear unit (ReLU) [4] is used as the activation function, which is obtained as

$$f_a(z) = \max(0, z) \tag{2.10}$$

The output from the output layer is the prediction targets, $\{\hat{y}\}$. In this research, three CTLE adaptation results are the model prediction outputs. The weights, W^h, and bias, b^h, of each layer are needed to be trained. The stochastic gradient descent (SGD) method [5] is used to minimize the cost function, which can be represented as

$$W^h = W^h - \gamma \widehat{\nabla}_{W^h, b^h} E \quad \forall h = 1, \dots n, out \tag{2.11}$$

where $\widehat{\nabla}_{W^h, b^h} E$ is the stochastic approximation to the true gradient. γ is the learning rate during the training process. The stochastic gradients are computed by the backpropagation method.

2.4.2. Long short-term memory (LSTM) model

As a state-of-the-art deep learning architecture is designed for time-series regression problem. RNN is widely used in forecasting such problem [6]. The RNN aims to map the input sequence x into outputs y. Each output in the sequence is calculated by the state of the previous RNN cell and current input. To reduce the training complexity, the structures of each RNN cells are the same. The unfolded topological graph is shown in Figure 2.11, which can demonstrate the working process of RNN.

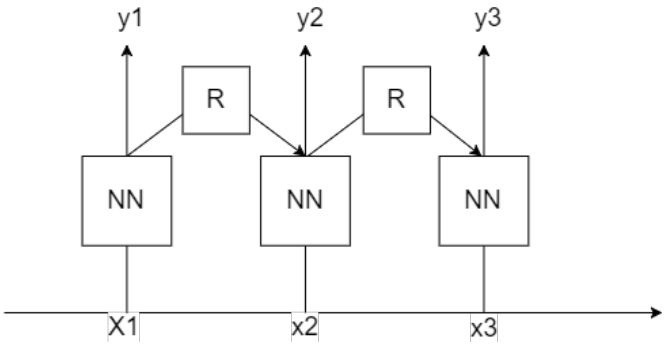


Figure 2.11. RNN unfolded topological graph

Traditional RNN or Vanilla RNN (VRNN) structure faces a challenge. If the training length of RNN is significantly long and non-truncated backpropagation is applied, due to a vanishing gradient or exploding gradient, backpropagated errors will get smaller or larger layer by layer, which makes backpropagation insignificant. To deal with the vanishing gradient or exploding gradient problems, the LSTM model is proposed [7]. The LSTM cell structure is shown in Figure 2.12. LSTM can create paths where the gradient can flow for a long duration.

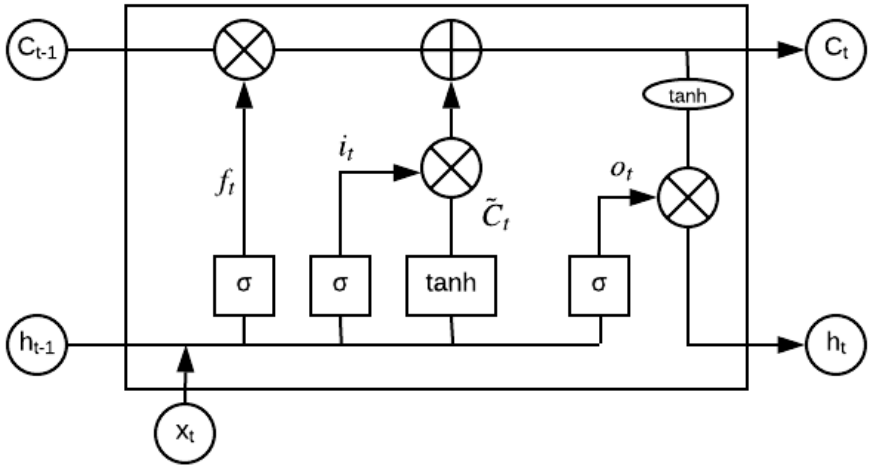


Figure 2.12. LSTM cell structure

The core function of the LSTM is the cell state, which is the horizontal line on the top of Figure 2.12. The cell state would go through the entire chain, with some linear interactions. The LSTM has the capability to remove or add previous or new information to the cell state using gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural

networks layer and a pointwise multiplication operation. An LSTM cell has three gates, namely input, output and forget, to protect and control the cell state. The input gate would add new information selected from the current input and previous sharing parameter vector into the current cell. The forget gate is to discard useless information from the current memory cell. And the output gate decides new sharing parameter vector from the current memory cell.

At first, LSTM would discard useless information from the cell state. This decision is made by a sigmoid layer named the forget gate layer. It looks at h_{t-1} and x_t, and outputs a number between zero and one for each number in the cell state C_{t-1}. If the output is one, it means the LSTM would completely keep the cell state, while zero represents the LSTM would completely forget this. The forget gate can be calculated as

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.12}$$

where f_t is the output of the forget gate. W_f and b_f are the weights and biases of the neural networks. h_{t-1} is the previous hidden vector. x_t is the current input.

The next step is to decide what new information is needed to store in the cell state. This step is twofold. First, a sigmoid layer named the input gate layer, decides which values will be updated. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, which could be added to the state. Second, these two will be combined to create an update to the state. The input gate can be represented by

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{2.13}$$

where i_t is the output of the input gate layer. W_i and b_i are the weights and biases of the input gate layer. $\tilde{C}_t$ are the vectors which modify the cell state.

Then, the old cell state, C_{t-1}, will be updated, as shown below

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.14}$$

where C_t is the new cell state. f_t and i_t are the outputs of the forget gate and input gate respectively. These two gate outputs will decide how much information will be through away and updated, which can be obtained as

$$\begin{aligned} f_t &= \sigma\big(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\big) \\ i_t &= \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o) \end{aligned} \tag{2.15}$$

Finally, the LSTM cell will decide what should be considered as the final output. A sigmoid layer will decide what parts of the cell state are the output. Then, the cell state, C_t, will go through a tanh layer and multiply it by the output of the sigmoid gate. The output of the LSTM can be obtained by

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{2.16}$$

where o_t is the output of the output gate. h_t is the final output of the LSTM. W_o and b_o are the weights and biases of the output gate layer.

With these three gates, the LSTM model will remember all the useful information from the receiver input and void gradient vanishing or exploding problems.

Chapter 3. State-of-the-Art

Recently, numerous works have been done in high-speed receiver behavioral modeling to speedup post-silicon validation [8]. In this chapter, we would present the state-of-the-art receiver behavioral modeling using machine learning approach. The high-speed receiver behavioral modeling can be divided into two areas: 1) receiver transient behavior modeling and 2) receiver adaptation modeling.

3.1. Receiver transient behavioral modeling

The receiver transient behavior modeling has two modeling methods: grey-box modeling method and black-box modeling method. The grey-box modeling methods are used to predict only eye height and eye width. The black-box modeling methods would predict the receiver transient output waveforms.

As for the eye margin prediction problem, in [9], a grey-box modeling method is presented. They took a deep look into their circuit, extracted 10K features and selected 30 features, using feature selection method, to predict eye height and eye width at the receiver. They did a classification and regression prediction of eye margin. As for eye margin classification prediction, they set a threshold and used couples of machine learning models, like Random Forest, Boosted Trees, to predict whether the eye is passed or failed. As for eye margin prediction, they used machine learning regression models, like Random Forest, SVM, to predict eye height and eye width respectively. The results showed that all the model prediction accuracies are low. The worst-case testing accuracy is only 87%. Because it's a grey-box modeling method, they need some SerDes experts to handle parameters in the receiver and examine whether the selected features are reasonable. [10] used Artificial Neural Networks (ANN) to predict eye width and eye height after the receiver equalization with a small amount of testing and training data and possibly a large

number of knobs. They tested their methodology on two current industrial high-speed channel topologies, namely USB3 SuperSpeed Gen1 and SATA Gen 3. It's a grey-box modeling method because they need to use domain knowledge to select 7 and 10 important receiver knobs for USB3 SuperSpeed Gen1 and SATA Gen 3 respectively. Using a system response sampling strategy, their model accuracies are around 95% for eye width and eye height prediction. However, their receiver model doesn't include DFE, which is a non-linear component in the receiver. The DFE is commonly used in most of the high-speed SerDes systems. Meanwhile, it's not practical to only predict eye width and eye height in the SerDes system. [12] uses the DNN model to predict eye width and eye height using 8 human-selected features in the SerDes channel. Again, their receiver model doesn't include the DFE block, and their model can only predict eye width and eye height. [28] and [29] presented SVM models to predict the eye height and eye width using five selected features. The prediction results didn't show a very high correlation.

The eye width and eye height measurement only contain four data points in an eye diagram, which cannot be used to plot bathtub curves. The bathtub curve shows the horizontal or vertical eye-opening at each bit error rate (BER) level. It is based on the jitter probability density function, which can be only extracted from an eye diagram.

In a SerDes system, there are approximately thousands of circuit features impacting the recovered signal in SerDes receiver. The published grey-box methods take a couple out of the thousands of features to build the ML model and do prediction. The tradeoff in the grey-box modeling method is the more features adopted in the ML model the more accurate the prediction but the model complexity and simulation time increase. The accuracy of the grey-box method will be sacrificed to balance the model complexity as well as simulation time.

As for the transient waveform prediction, [14] uses a system identification modeling method to predict the receiver output waveforms. However, they didn't validate their model using eye diagrams and only a few bits are tested. [11] and [12] presented a black-box modeling method to predict transient waveforms at the receiver, which can be used to plot eye diagrams. Deep learning models namely stacked Recurrent Neural Networks (RNN) and stacked Long Short-Term Memory (LSTM) model, are trained based on PRBS data. However, the current black box training methods can only predict the transient waveforms from the same PRBS data set and the same channel that the training data are from. But they did not provide any information about waveform prediction accuracy. And they need continuous training transient waveforms to feed into their model to obtain high accuracy. Meanwhile, either the RNN or LSTM model has very long training time, normally several hours or even a day, with a very-long-bit PRBS data sequence. What's more, due to their model accuracy limitation, the eye diagram generated from predicted transient waveform shows a low correlation with the real eye. Also, their test data are less than 100 UIs, which are not enough to test their model accuracy. As for signal integrity analysis, they only did the eye diagram comparison.

According to the recent works, there are no modeling methods which can only train the machine learning model once and predict the transient waveforms, not from the same training PRBS data and training channel. Moreover, some publications provide prediction accuracy while some do not. The provided accuracy numbers are all about eye height and eye width prediction without giving the accuracy number for predicted transient waveforms.

Nowadays, no publications are found to do bathtubs prediction. The bathtub prediction is based on the eye diagram constructed from long enough transient waveforms. Generally, it needs at least half-million bits. The ML model complexity determines the model prediction/simulation

time and consequently, the total length of the transient waveform prediction can be obtained with reasonable simulation time. All the latest published work can predict much fewer bits required for bathtub prediction.

The proposed modeling method in this work provides fast and high-precision simulation results for the transient waveform, eye diagram and bathtub curve.

3.2 Receiver adaptation modeling

With the transceiver becoming more and more complex, it's impossible to tune these parameters manually. In that case, equalization adaptation techniques for the on-die circuit are proposed [36] [37] [38] [39] [40]. An adaptive tuning approach allows the optimization of the equalizers for varying channels, environmental conditions, and data rates. One of the commonly used CTLE adaptation technique is CTLE tuning with output amplitude measurement [37]. As for the DFE adaptation, [39] proposed 2x oversampling the equalized signal at the edges can be used to extract information to adapt a DFE and drive a CDR loop, and sign-sign least mean square (LMS) algorithm used to adapt DFE tap values.

Nowadays, only IBIS-AMI model has the capability to provide the adaptation simulation [15]. To find the best receiver coefficients, LMS adaptation methods are used. The LMS algorithm is an approximate gradient descent optimizer [53]. Its objective function is the mean squared error between the equalizer output and the desired equalizer output, which is assumed to be the transmitted waveform $d(t)$, sampled at the center of the eye pattern. Letting $e(kT) = d(kT) - r(kT)$, the mean square error is

$$\xi = E[e^2(kT)] \tag{3.1}$$

The approximation comes in using the instantaneous value of the squared error as a noisy estimate of its expected value. A constant μ is defined to control the adaptation's rate of convergence. The result is the familiar LMS update equation

$$\vec{p}(kT) = \vec{p}(kT - T) + 2\mu e(kT)\vec{\mu}(kT) \tag{3.2}$$

Hence, the LMS algorithm would minimize the mean squared error and find the best receiver settings.

However, the IBIS-AMI model has some limitations. The IBIS-AMI model can only do adaptation and transient simulation at the same time, which would be inefficient in two scenarios. The first scenario is that users want to get transient simulation results to see the eye-opening or bathtub curve information. The second scenario is that users only need to know the receiver adaptation codes, which represent the equalization status of the SerDes link. In that case, we need to find a way to build behavioral models for these two simulations separately, while the simulation speed of the behavioral models should be faster than the IBIS-AMI model.

To the best of our knowledge, there is no work focusing on the SerDes receiver adaptation behavioral modeling using the machine learning approach. The reason why no one touches this area is that the response of the system is dynamic. In the LTI system, the response of the system is linear and won't change during the observation. The adaptation process in the high-speed SerDes link is a non-LTI system, which is difficult to build a behavioral model.

In this work, the proposed modeling mechanism can show a high correlation with the receiver adaptation codes in the on-die circuit. Meanwhile, the simulation speed of the proposed modeling method is much faster than the IBIS-AMI model.

Chapter 4. Receiver transient behavioral modeling

It's crucial to build the SerDes behavioral models with high simulation speed and good hardware correlation. The current SerDes simulation model, IBIS-AMI model, takes thirty minutes to do the transient simulation. In this chapter, we focus on building a high-speed SerDes receiver equalized transient behavior model, which has a fast simulation speed and excellent correlation.

At first, we explored various machine learning model types using the data collected from a 10 Gbps Avago transceiver and select the best model type for the transient modeling. Second, a different transceiver, Xilinx UltraScale+ GTY transceiver, is used to collect data and test the robustness of the selected model type. Xilinx UltraScale+ GTY transceiver is a 100 GE application, which the data rate is 25Gbps. To further improve model performance, an adaptive-ordered system identification model is proposed to mimic the behavior of the high-speed receiver. Transient waveform, eye diagram and bathtub curve are used as the model performance metrics.

4.1. Model type exploration

At the beginning of this research, a 10 Gbps Avago transceiver is used to collect data. Next, how to collect the data and do the model training are briefly introduced.

4.1.1. Data collection

Figure 4.1 shows how to measure the receiver input and output using the scope. To collect various cases, different data pattern, TX settings, backplane channels, and RX settings are swept. The data configuration of each component in the SerDes link is shown in Table 4.1.

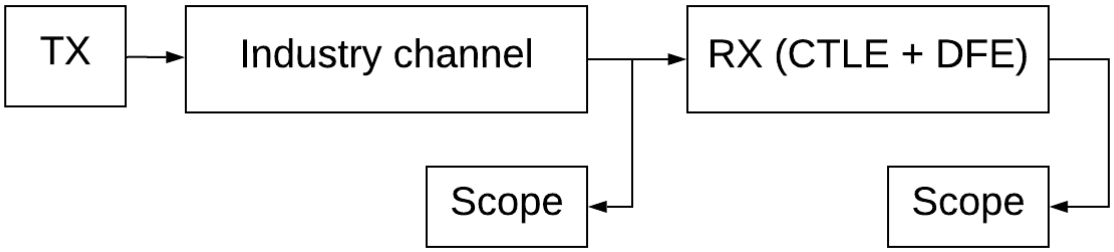


Figure 4.1. Data measurement from the 10 Gbps Avago transceiver

Table 4.1. Avago transceiver setting configuration

	configurations
Data rate	10 Gpbs
Data Pattern	PRBS7
Channel insertion loss	Low, medium, high
TX settings	Low, medium, high swing
RX CTLE Gain	Low, medium, high
DFE	Off, on

During the data collection, the receiver equalization setting is fixed, and the system response is stable. Our goal is to build a receiver behavioral model under a fixed receiver setting.

In this experiment, because there are 3 different channels for the Avago chip, CTLE and DFE settings are tuned to make eye open reasonably large for different channels. CTLE settings include DC gain, low-frequency gain and high-frequency gain. As for the DFE, the first two DFE tap values are considered and other DFE taps are set as zero gain. In this experiment, 3 CTLE and 2 DFE settings are tuned under three different channels. The criterion is to make sure it would pass the eye mask test at the receiver for each measured case, which ensures that the signal-to-noise ratio is large enough. Because machine learning models can only learn the receiver equalization

behavior but not random noise. If the signal-to-noise ratio is small, the model predicted accuracy would drop because of the random noise. In this experiment, the receiver input and output waveforms for 50 different CTLE and DFE settings cases are measured. Those 50 receiver settings are from three different channels. Both receiver input waveform and output waveform are measured using the real-time oscilloscope. The receiver input and output waveform would be the input and output of the model. The number of measured input bits and output bits are 7128. The data rate is 10Gbps. As for the data measurement, there are 16 sampling points per UI.

4.1.2. Model performance comparison

There are 5 different model types considered in this work. Recall in Section 2.2, system identification models can be divided into two types, linear system identification modeling, and nonlinear system identification modeling. In the linear system identification modeling, ARX and ARMAX model are considered in this work. As for the nonlinear system identification modeling, NNARX model is used. RNN model, which is often used in time-series data prediction problem, is also used in this research.

As for the model accuracy validation, we use the normalized root mean square accuracy (NRMSA). This method is commonly used in time series data prediction problem. The NRMSA can be calculated as

$$NRMSA = 100 \times \left[1 - \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}}\right] \tag{4.1}$$

where y_i is the real output at time i. $\hat{y}_i$ is the model predicted output at time i, and $\bar{y}$ is the mean of all the real outputs. If the predicted output is close to the real data, the NRMSA is close to 100 scores.

As for the system identification models, the input and output orders mean how many previous inputs and outputs should be considered to predict the current output. To simplify the model process, we set the input order equal to the output order in the system identification model. If only a few past inputs and outputs are considered, which means the orders of input and output are low, the model accuracy would be low. On the other hand, if too many past inputs and outputs are chosen, which means the orders of input and output are high, the model would be more complex and training time increases. Choosing the best trade-off point is the key to building model process. Order sweeping method is used for selecting the best order of each model.

In this experiment, the order sweeping method is used to select the best orders of the ARX, ARMAX, and NNARX model. What's more, to reduce the sweeping iteration, the order of inputs and outputs of the models are set as the same.

In this work, for nonlinear system identification models, twenty hidden neurons with tanh activation function in one hidden layer and one linear output neuron are set in neural networks architecture. For the LSTM model, the batch size is set to 100. 200 hidden units are used in each LSTM cells. The learning rate is 0.004.

Figure 4.2 shows prediction results of one case for linear system identification models, nonlinear system identification models, and LSTM model respectively. This is the case that only CTLE works.

As for linear system identification models, ARX model shows low correlation with the real data. ARX model cannot predict the low and high frequency gain accurately. On the other hand, the performance of the ARMAX model is a little better than ARX model. It can predict the high frequency gain accurately, but not the low frequency gain. The worst accuracies for the ARX and ARMAX model are 76.34% and 80.62% respectively.

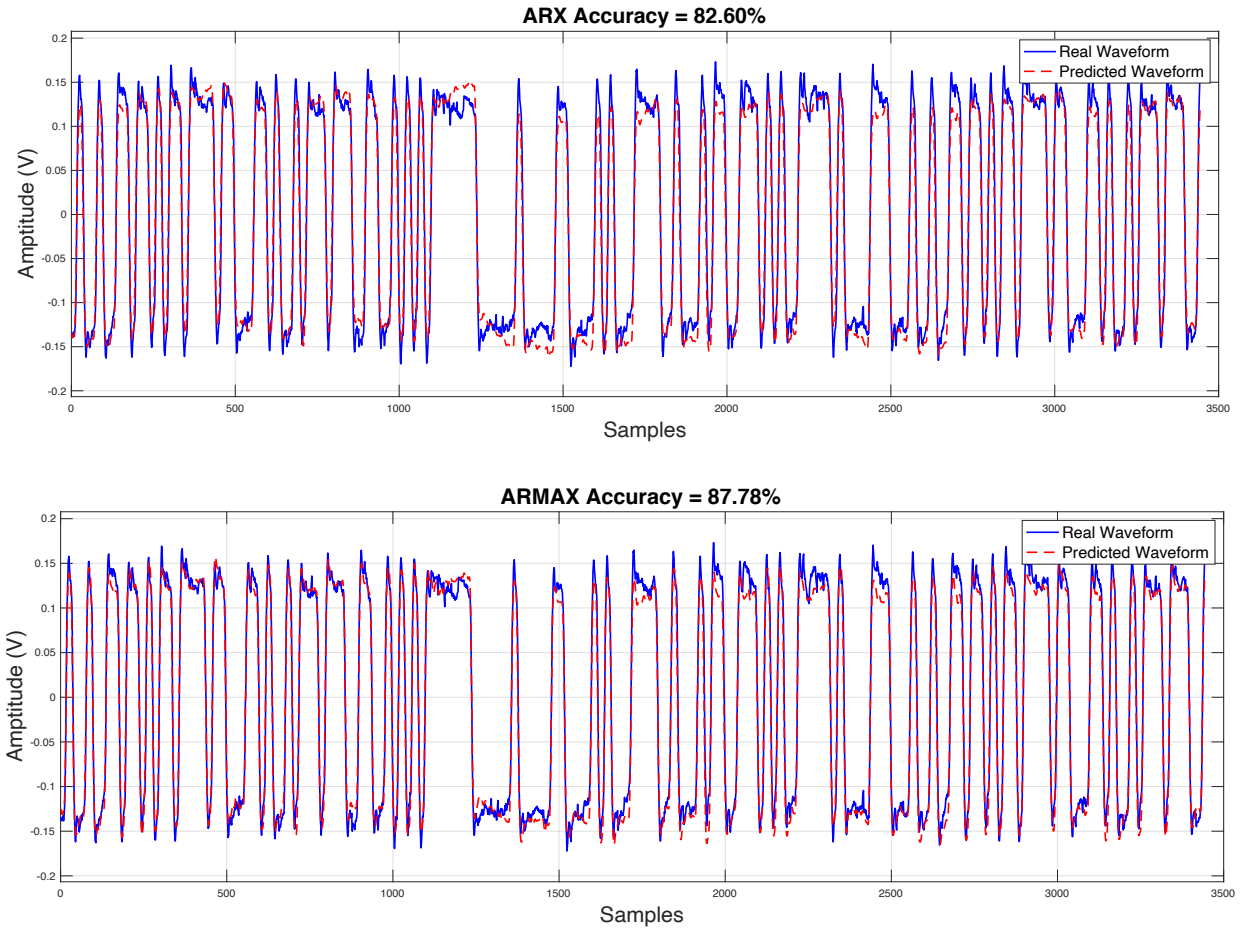


Figure 4.2. Linear system identification model performance

As for nonlinear system identification model, shown in Figure 4.3, the NNARX model shows high correlations with the real data. Both models can predict the low and high frequency gain accurately. The worst accuracies for the NNARX model is 93.11%.

As for the LSTM model prediction in Figure 4.4, it also shows a high correlation with the real waveform. The LSTM model can predict the low and high frequency gain accurately. The worst accuracy for the LSTM model is 93.07%. However, the prediction accuracy of the LSTM model is lower than the nonlinear system identification models. What's more, the training time of the LSTM model is much longer than system identification models.

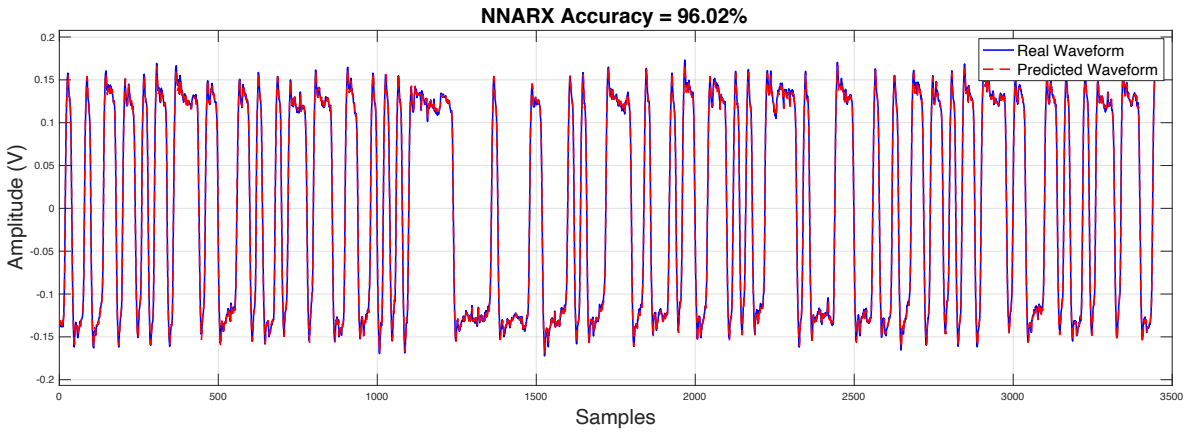


Figure 4.3. Nonlinear system identification model performance

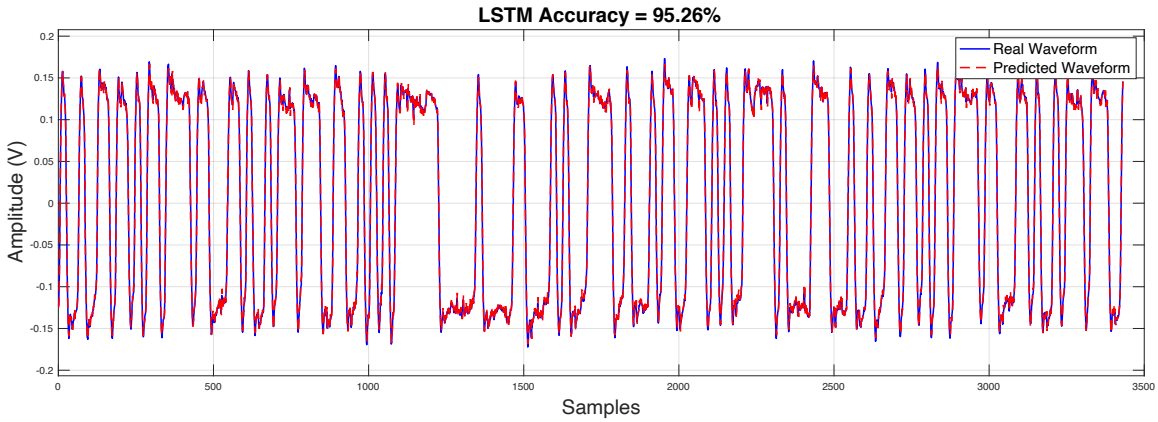


Figure 4.4. LSTM model performance

The model performances for all the 50 cases are shown in Figure 4.5. For linear system identification models, the ARMAX model shows better performance than ARX model. The orders the ARX and ARMAX models are 20, which means previous 20 sampling points of input and output data are used to predict the current output. The orders the NNARX models are 10. Nonlinear system identification model and the LSTM model have similar model performance. However, the average prediction accuracies of the LSTM model and NNARX model are 95.71% and 95.78% respectively. What's more, the average worst case accuracies of the LSTM model and NNARX model are 93.23% and 92.96% respectively. The performance of the NNARX model is better than the LSTM model.

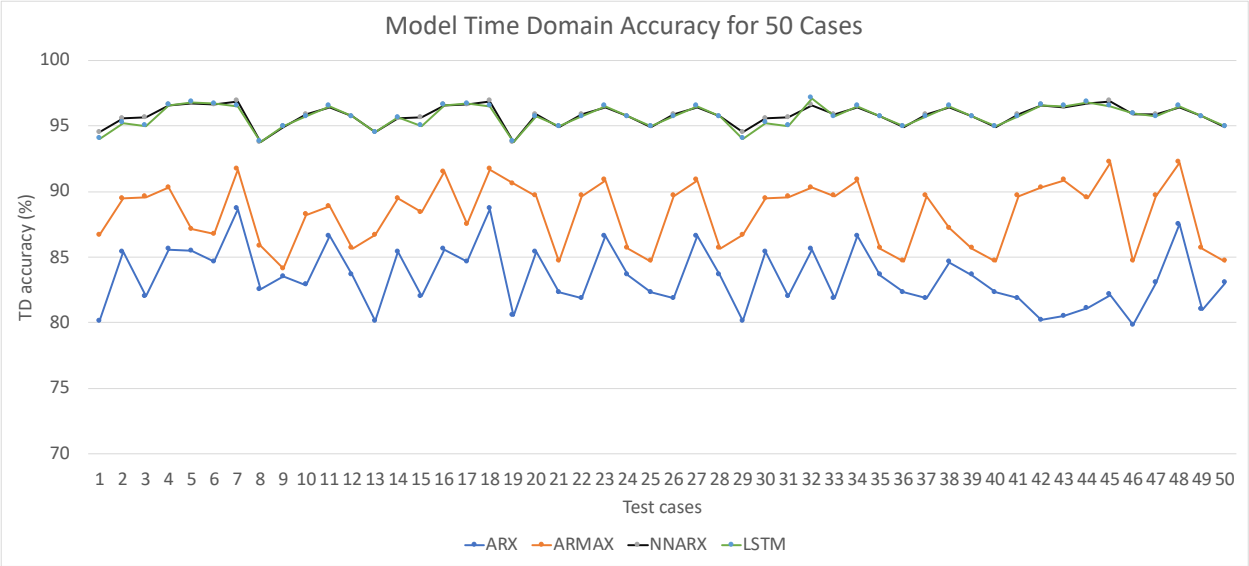


Figure 4.5. Model performances for 50 cases

As for the transient simulation, the simulation speed of the model is also important. The IBIS-AMI model takes 30 minutes to simulate half million bits. For the linear and nonlinear system identification models, due to its simple model structure, all the models need around 10 seconds for half million-bit simulation. However, it takes about 5 minutes for the LSTM model simulation.

Here are the reasons why the NNARX model's performance is better than the LSTM model:

1) To overcome vanishing gradient problem, the LSTM model have three gates, which can bypass units and remember for longer time steps [18]. Since the LSTM model can remember information happened very long time ago, too much unrelated information is considered to predict the current output. The advantage of the LSTM model is suitable in the sequence prediction but not in the receiver behavioral modeling because the current output is only influenced by the recent previous bit due to the channel loss.
2) When the DFE is involved, the current output depends on previous bit values. How many DFE taps the receiver has decides how many previous bit values are going to effect on the current bit value. The LSTM model can remember previous information

long time ago, which doesn't match the DFE feedback structure. On the other hand, the memory depth of the nonlinear system identification is the order of the input and the output, which set by users (adaptive-ordered system identification model will be introduced in the next Section). In that case, only a few recent information would be used, in the nonlinear system identification model, to predict the current output, which increases the correlation among the model inputs and outputs.

3) The LSTM model as a complex the model structure, which makes the training time become slower than other models. On the contrary, the nonlinear system identification model has a simpler structure.

Considering the model performance and simulation speed, the NNARX model is the best model type for the receiver equalized transient behavioral modeling.

4.1.3. NNARX model training and prediction process

The NNARX model is a recurrent dynamic network, with feedback connections enclosing several layers of the network. The NARX model is based on the linear ARX model, which is commonly used in time-series modeling.

The defining equation for the NARX model is:

$$y(t) = f(y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u)) \tag{4.2}$$

where the next value of the dependent output signal y(t) is regressed on previous values of the output signal and previous values of an independent (exogenous) input signal. In the NNARX model, function f is a feedforward neural network. A diagram of the NNARX model training process is shown in Figure 4.6 (a). Since the true output is available during training, the true output itself can be used instead of feeding back the estimated output. This will have two advantages. The first advantage is that the input to the feed forward network is more accurate. The second is that

the static back propagation can be used for training instead of dynamic back propagation, which has complex error surfaces exposing the network to higher chances of getting trapped in local minima and hence requiring a greater number of training iterations.

Once the outputs have been calculated, the NNARX model would compute the error E, which is defined by the expression:

$$E = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.3}$$

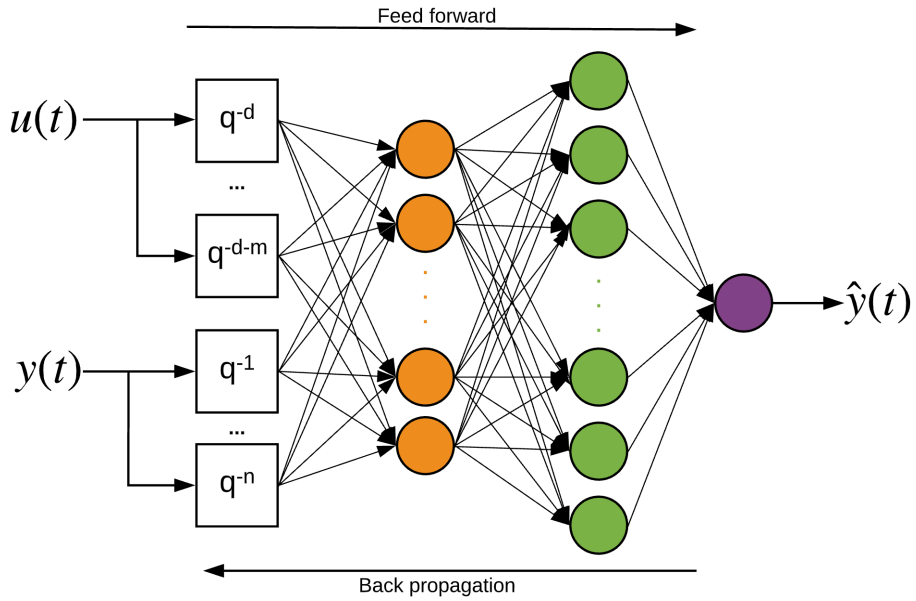
where $\hat{y}_i$ is the model predicted output and y_i is the desired output. The most commonly used learning algorithm is the gradient descent algorithm. In this the global error calculated is propagated backward to the input layer through weight connections as in Figure 4.6 (a). During the back-propagation process, Levenberg-Marquardt algorithm [51] is used in the model, which can be written as

$$W_{new} = W_{old} - [J^T + \gamma I]^{-1} J^T E(W_{old}) \tag{4.4}$$

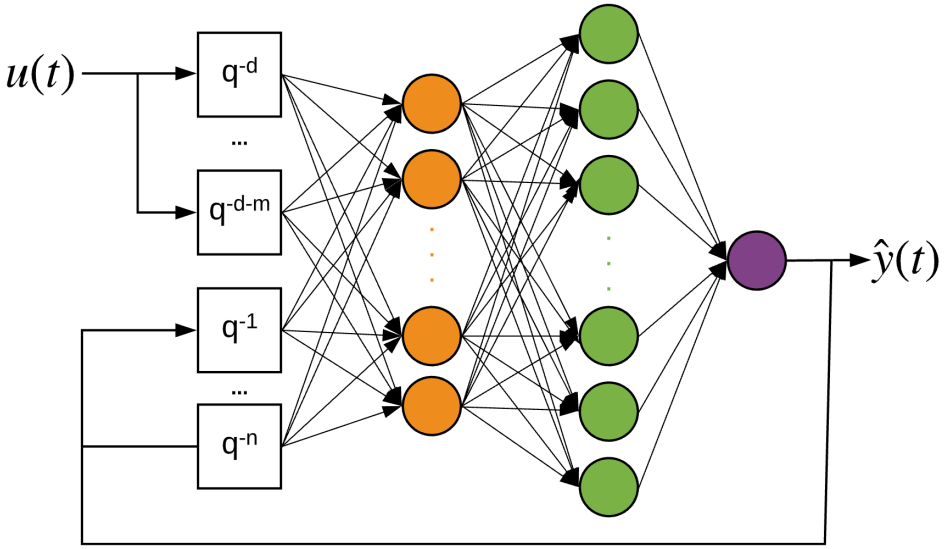
Where J is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights and biases, I is the identity matrix and γ is the parameter used to define the iteration step value. It minimizes the error function while trying to keep the step between old weights (W_{old}) and new updated weights (W_{new}).

After the NNARX model is well-trained, the prediction process of the NNARX model is shown in Figure 4.6 (b). The next value of the dependent output signal y(t) is regressed on previous values of an independent (exogenous) input signal. The output of NNARX network can be considered to be an estimate of the output of some non-linear dynamic system that is being modeled. The output is fed back to the input of the feed forward network in the NNARX

architecture. In this way, the NNARX model can predict the current output based on previous inputs and predicted outputs.



(a) NNARX model training process



(b) NNARX model prediction process

Figure 4.6. NNARX model training and prediction process

4.2. Improved model for the receiver equalized transient behavioral modeling

To test the robustness of the NNARX model, a different technology is used. In this work, the data was collected from the on-die measurement from Xilinx UltraScale+ GTY transceiver.

4.2.1. Data Collection

This work focuses on 100 Gigabit Ethernet (100GE) application [20] [35]. In the 100GE application, there are 4 lanes and 25G each. Specifically, the data rate is 25.78Gbps.

The data was collected from the Xilinx UltraScale+ GTY transceiver [41] [42]. In the Xilinx UltraScale+ GTY transceiver, there are 3 CTLE stages and 15 DFE taps.

Table 4.2. UltraScale+ GTY transceiver setting configuration

	Configurations
Data rate	25.78 Gpbs
Data Pattern	PRBS7, PRBS 15, PRBBS 23, PRBS 31
Channel insertion loss	Low, medium, high
TX	Low, medium, high swing
RX CTLE Gain	Low, medium, high
RX DFE tap value	Low, medium, high

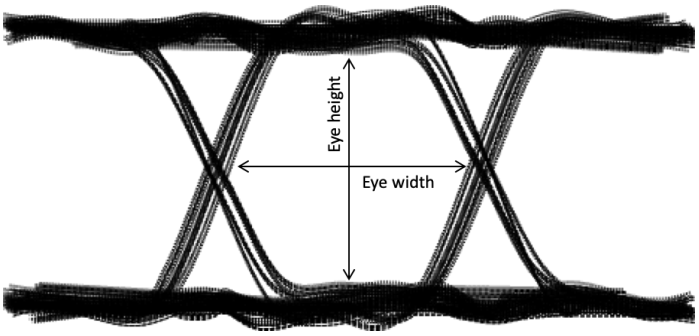
The data configurations are shown in Table 4.2. The UltraScale+ receiver has two modes, CTLE mode and DFE mode. The receiver will be switched to CTLE mode under very short reach (VSR), and short reach (SR). Those channel insertion losses are around -15dB (at Nyquist frequency). In the CTLE mode, DFE is bypassed and only CTLE works. Because CTLE is a linear block, linear model will be used in CTLE mode. The receiver is switched to DFE mode when it's under a medium reach (MR) or a long reach (LR). The insertion loss of the long reach is normally

between -22 dB to -30dB. In DFE mode, both CTLE and DFE will work together. In this work, both CTLE mode and DFE mode are modeled. Each time, the receiver input waveform and output waveform are collected using the oscilloscope.

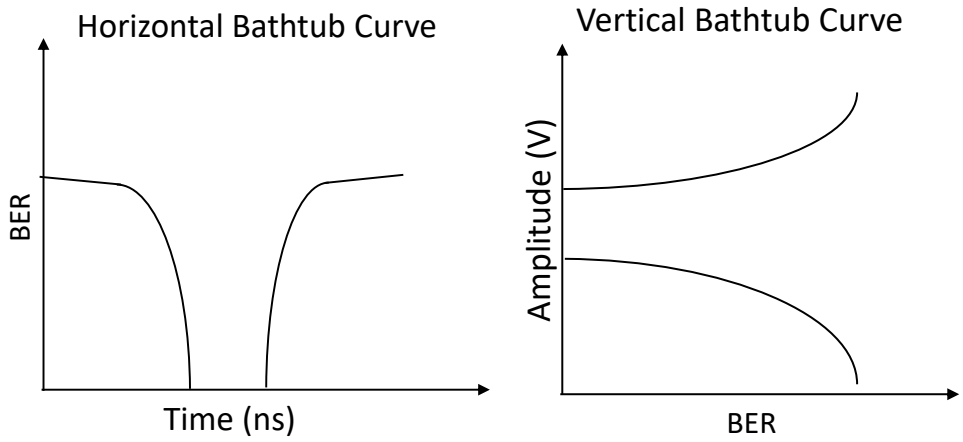
4.2.2. Signal integrity analysis

The main goal of the transient waveform simulation is to do the signal integrity analysis. As for high-speed SerDes design evaluation, two diagnostic tools in signal integrity analysis, eye diagram and bathtub curve, are most commonly used after the receiver equalization. Eye diagrams, as shown in Figure 4.7 (a), are generated by overlaying time-domain waveforms corresponding to signal amplitude, represented by the vertical dimension, and time, represented by the horizontal dimension. The traces are superimposed in persistence mode showing the envelope of amplitude and timing fluctuations. The region in the center that is devoid of any traces typically resembles an eye. The eye diagram can qualitatively show the range of amplitude and timing deviations associated with the data. An eye diagram with a large eye opening is indicative of a data stream that has very little amplitude noise and timing noise. An eye diagram with a small opening indicates that the data is very noisy.

Bathtub curve, shown in Figure 4.7 (b), is a statistical measurement of the eye diagram. The horizontal and vertical bathtub curve can display the eye width and eye height at each BER level. The bathtub prediction is based on the eye diagram constructed from long enough transient waveform, normally from 1e+5 UIs to 1e+7 UIs in the industry. Bathtub curves are created from the jitter probability density function (PDF) and extrapolation. The eye diagram contains all the information of total jitter PDF, which is a convolution integral of bounded deterministic jitter (DJ) and unbounded Gaussian random jitter (RJ).



(a) eye diagram



(b) bathtub curve

Figure 4.7. Two signal integrity analysis methods

For example, if we capture 1e+5 UIs and generate an eye diagram, the horizontal and vertical bathtub curve, before 1e-5 BER level, can be plotted accurately according to the integral of the jitter and noise PDF respectively. The tails of the horizontal and vertical bathtub curve are determined by Gaussian random jitter and random noise, which may or may not be accurate. If we wait long enough and capture 1e+12 UIs, the eye diagram would become more real, and we can confidently plot the bathtub curve down to 1e-12 BER level. With longer UI information, we can plot a more accurate bathtub curve which match the real system.

Nowadays, all the researches focus on the transient waveform and eye margin prediction using machine learning (ML) models. In their models, only short-length data, for example, 100 UIs, are predicted, which are not enough to generate high-correlation bathtub curves.

In this research, a machine learning based receiver behavioral model is built, which can quickly simulate transient waveform, eye diagram, and bathtub curve. This a fast and high-precision black-box modeling method, which will save lots of human efforts and validation time compared to the IBIS-AMI model. This method can predict any data patterns with the same accuracy level without re-training.

4.2.3. Improved system identification model

In Section 4.1, the NNARX model is proved to have the best performance. We would use this model type in this modeling. Previously, an order sweeping method is used to select the best order setting for the NNARX model. And the input order and the output order are the same for each model, which would limit the model performance. In this section, an improved system identification model, adaptive-ordered NNARX (ANNARX), is proposed to solve this problem.

Due to the channel loss, the current received bit is impacted by the ISI from the previous bits. This means the long-time-ago bits have very little effect on the current bit. So, during the model training process, our model doesn't need to remember the long-time-ago bit information.

As a result, Pearson Correlation Coefficient (PCC) is used to analyze the relationship among the current output, previous inputs and previous outputs. PCC is a measure of the correlation between two variables, the formula is shown as:

$$\rho_{X,Y} = \frac{cov(X, Y)}{\sigma_x \sigma_y} \tag{4.5}$$

where cov is the covariance, σ_x is the standard deviation (SD) of X, σ_y is the SD of Y.

The PCC scores for one case are shown in Figure 4.8. In this model, the threshold is set to 0.6, which is a common value for the PCC analysis [52]. If the PCC score is above 0.6, it's often considered as a strong correlation.

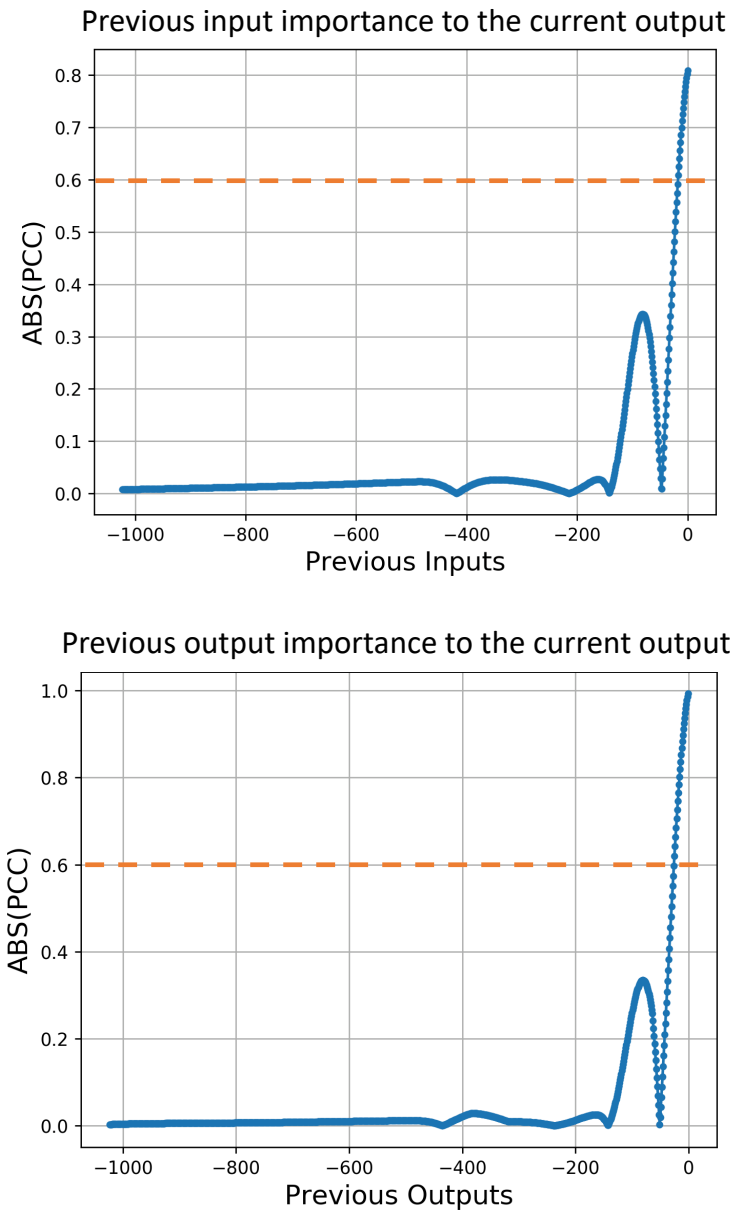


Figure 4.8. PCC scores of previous inputs and outputs for the current output

As well known that the LSTM model has a long memory length [7], which cannot be parameterized. On the contrary, system identification models, like NNARX, have limited memory

length, which can be defined by users. From this perspective, system identification models are more suitable for the receiver behavioral modeling. What's more, system identification models have shorter training time compared with the RNN [17] or LSTM model.

In this research, to identify the different effects of previous inputs and outputs, the adaptive-ordered system identification model, ANNARX, is proposed, as shown in Figure 4.9. During the training process, to identify the different effects of previous bits, the ANNARX model will analyze inputs and outputs and self-select how many previous inputs and outputs are needed to predict the current output according to PCC scores. The PCC threshold is set to 0.6, which is a common value for the PCC analysis [52]. The ANNARX model would analyze how many previous inputs and outputs are most related to the current output. After that, the ANNARX model would know the best system identification model order for the current receiver setting. Next, the ANNARX model would use the input data and ground-truth output data to train a NNARX model, which detailed training process can be found in Section 2.3.2. At the prediction process, the NNARX model for the current receiver setting can be used to simulate on new input waveforms under different channel or various TX settings. The predicted waveforms can be used to plot eye diagrams and bathtub curves.

Some detailed modeling scripts for the proposed model is shown in Appendix A.

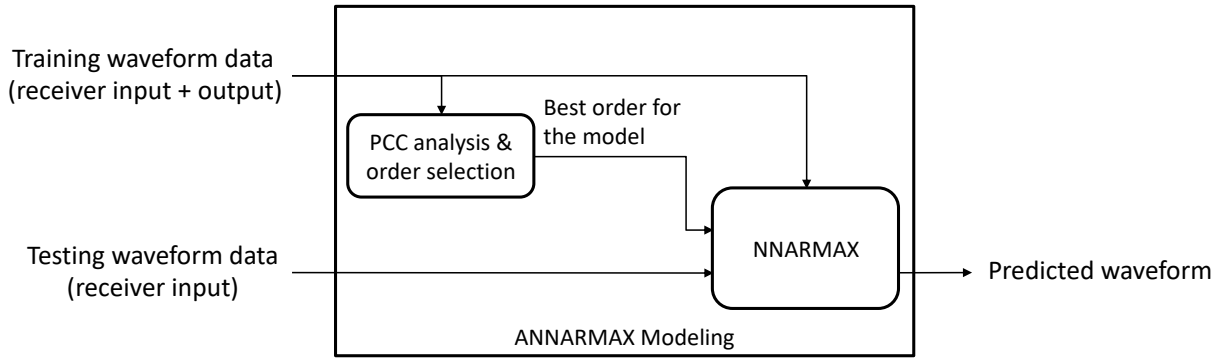


Figure 4.9. Adaptive-ordered system identification model structure

The proposed model can also be implemented in Verilog-A for use with a general-purpose circuit simulator. Verilog-A is a hardware description language (HDL), which is very commonly used in the industry.

To convert the ANNARX model to a Verilog-A model, the first thing is to re-write the PCC score from Equation 4.5 in HDL, which is easy to implement. As for the neural network structure, the feedback loop is converted from a collection of finite difference equations to a collection of differential equations; this will allow the circuit to be simulated using a different model order from that used for training. Then, the differential equations can be implemented as a Verilog-A model [43].

Once our model is converted into a Verilog-A model, which is a type of AMI model, the industry can use our behavioral model in the SerDes link simulation.

4.2.4. Modeling process

The modeling process chart is shown in Figure 4.10. At the training process in Figure 4.10, we measure the on-die transient waveforms at both the receiver input and sampled output. The machine learning model's complexity is automatically adaptive according to the data. The proposed modeling method only needs to train the model once and then predicts transient waveforms and eye diagrams of different data patterns over different channels.

After the model training, at the prediction process, the ML model is used to predict transient waveforms via different channels and with different data patterns. The proposed modeling method has fast training and simulation speed. The eye diagrams are then extracted from the predicted transient waveforms. All the latest published works could only predict a few bits required for eye diagrams to catch all the ISI impacts. The proposed model can quickly predict long bits and generate high-correlation eye diagrams and bathtub curves.

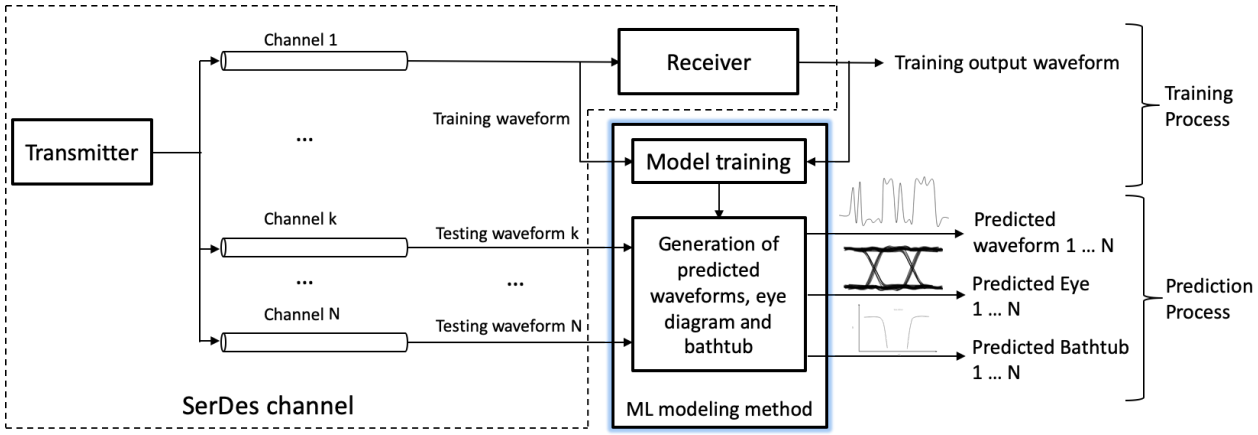


Figure 4.10. Receiver equalized transient modeling process chart

4.2.5. Proposed model simulation results

To test the model robustness, the training case is from one training channel and one data pattern. All the test cases are from different channels and different data patterns. Both CTLE mode and DFE mode are tested. The Xilinx UltraScale+ transceiver is a 16nm device, which has 3 CTLE codes and 15 DFE taps. The CTLE has three parameters, namely low-frequency gain (KL), high-frequency-gain (KH), and Auto Gain Control (AGC). The KL and KH are to boost low-frequency and high-frequency contents respectively after the channel. The AGC is to boost both low-frequency and high-frequency contents. In this experiment, 3 CTLE and 15 DFE settings are tuned under 6 different channels. The criterion is to make sure it would pass the eye mask test at the receiver for each measured case, which ensures that the signal-to-noise ratio is large enough. Because machine learning models can only learn the receiver behavior but not random noise. This criterion is also for the purpose of the model performance testing, because eye heights and eye widths are needed to be compared. If the ground-truth data has a closed eye, the predicted closed eye cannot be distinguished as correct prediction. However, with the proposed modeling method, the behaviors of one receiver under all the CTLE and DFE settings can be modeled correctly without any engineering insight or setting selection before modeling.

Table 4.3. Training and testing data configuration for CTLE mode and DFE mode

		Training	**Testing 1**	**Testing 2**
CTLE Mode	**TX setting**	Medium swing	Low swing	High swing
	Data pattern	PRBS 15	PRBS 15	PRBS 23
	Channel insertion loss	Low	Low	Medium
DFE Mode	**TX setting**	Medium swing	Low swing	High swing
	Data pattern	PRBS 15	PRBS 23	PRBS 31
	Channel insertion loss	High	Medium	High

For the CTLE mode, different TX settings and channels are used, as shown in Table 4.3. Under one particular CTLE and DFE setting, the signals at the receiver pass the eye margin test for three different TX settings and channels. The receiver setting is fixed during the measurements of the input and output data of the receiver for these three cases. At first, the model is trained on the data from a low insertion loss channel and a medium TX swing. After the ANNARX model is well trained, the one-time-trained model will be tested under two test cases. The first test case is from a low insertion loss channel and a low TX swing, and the other test case is from a medium insertion loss channel and a high TX swing.

Next, the two test case results are shown in Figure 4.11. The accuracy is calculated using the NRMSA method from Equation 4.1. The worst prediction accuracies for both cases are 99.3% and 99.2%. The proposed ANNARX model shows high correlation with the circuit transient behavior. The model can predict low and high frequency gain accurately. Besides the transient waveform comparison, prediction-generated eye diagram and bathtub curve are also compared with the real eye diagram and bathtub curve, which are shown in Figure 4.12 and Figure 4.13.

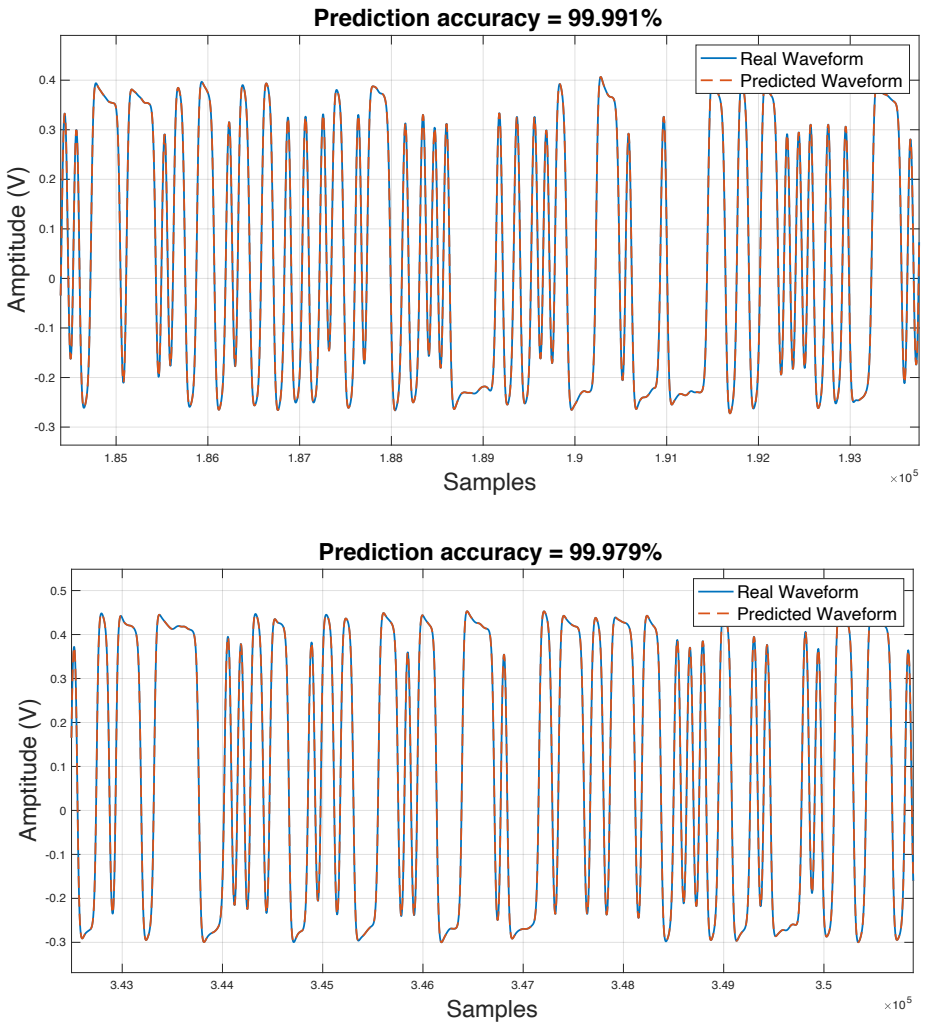
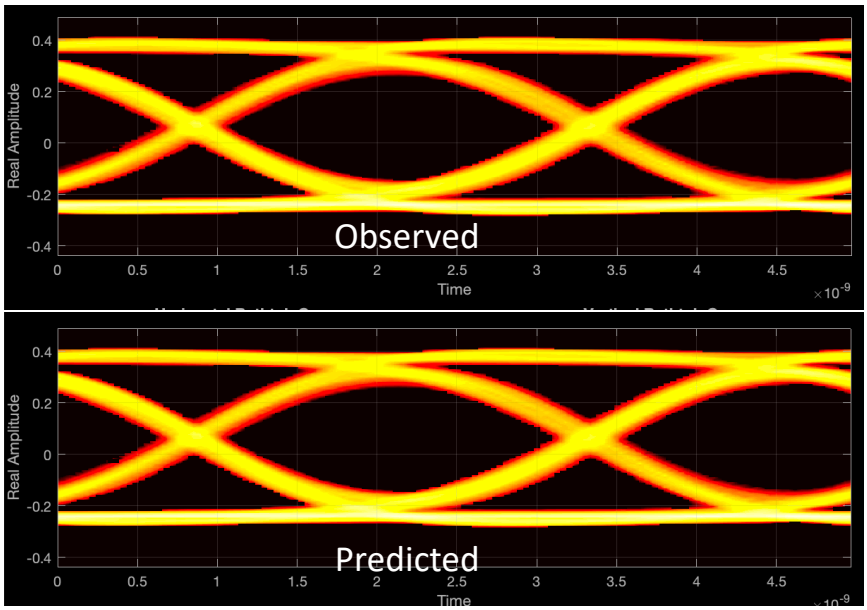
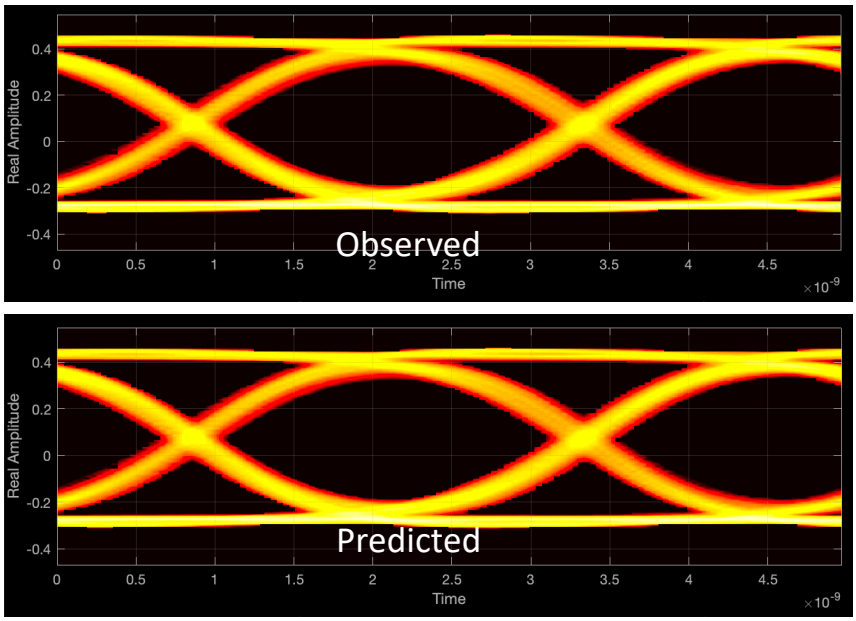


Figure 4.11. Waveform prediction results using the ANNARX model in CTLE mode

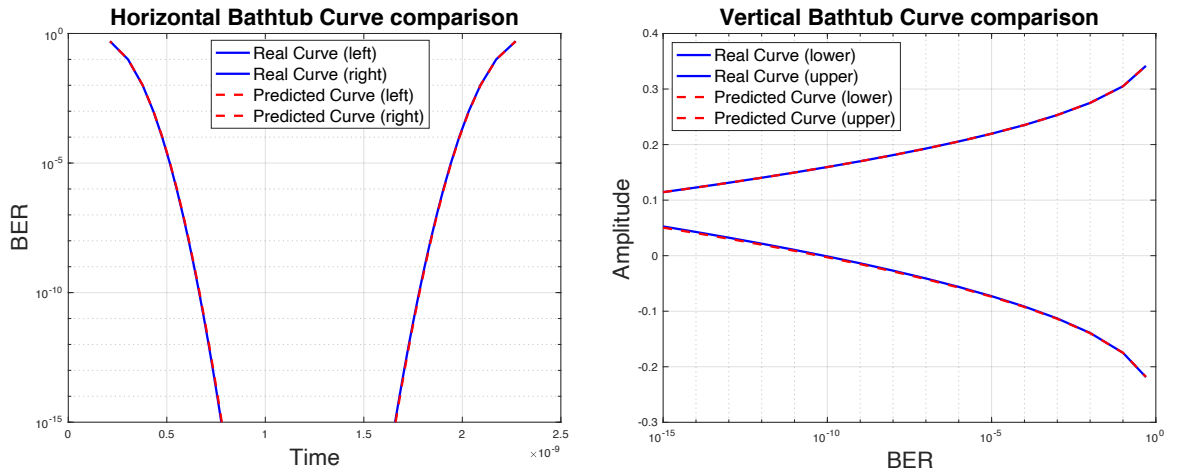


(a) Eye diagram comparison for test case 1

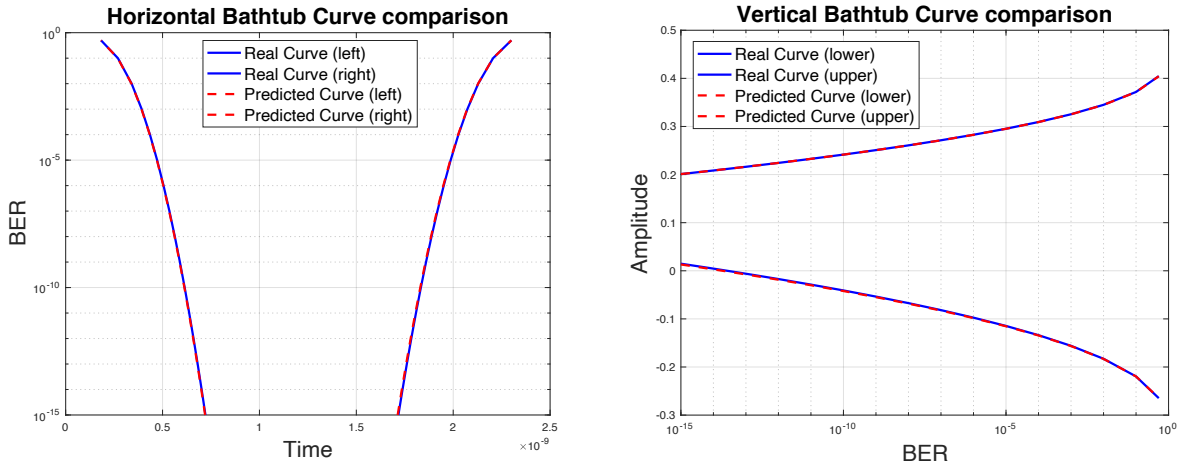


(b) Eye diagram comparison for test case 2

Figure 4.12. Real vs prediction eye diagram comparison in CTLE mode



(a) Bathtub curve comparison for test case 1



(b) Bathtub curve comparison for test case 2

Figure 4.13. Real vs prediction bathtub curve comparison in CTLE mode

In Figure 4.12, the eye diagram generated from the predicted transient waveform has a high correlation with the real eye diagram. The inner eye shapes are the same. Table 4.4 shows the eye height and eye width prediction results. The proposed ANNARX model can provide high-precision eye margin prediction. The prediction results are better than other related works [9] [10] [11] [12].

For the bathtub curve prediction in Figure 4.13, the bathtub curve generated from the predicted transient waveform has high correlation with the real curve horizontally and vertically. The RMSE and the worst error for different cases are shown in Table 4.5.

Table 4.4. Eye margin prediction results for CTLE mode

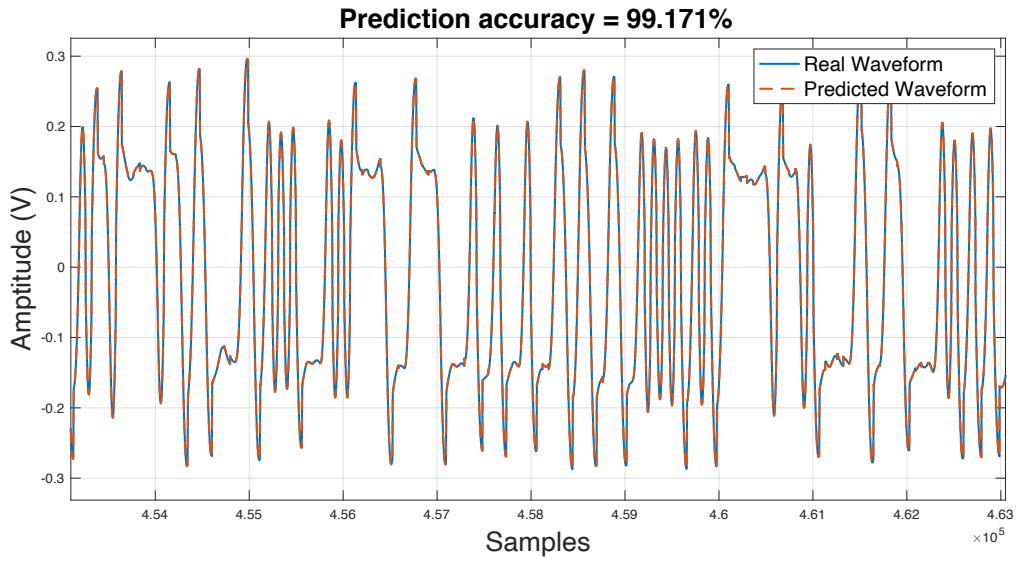
	Test case 1			Test case 2		
	Real	Prediction	Accuracy	Real	Prediction	Accuracy
Eye height (V)	0.37	0.37	100%	0.49	0.49	100%
Eye width (ns)	1.14	1.14	100%	1.32	1.32	100%

Table 4.5. Bathtub curve prediction results for CTLE mode

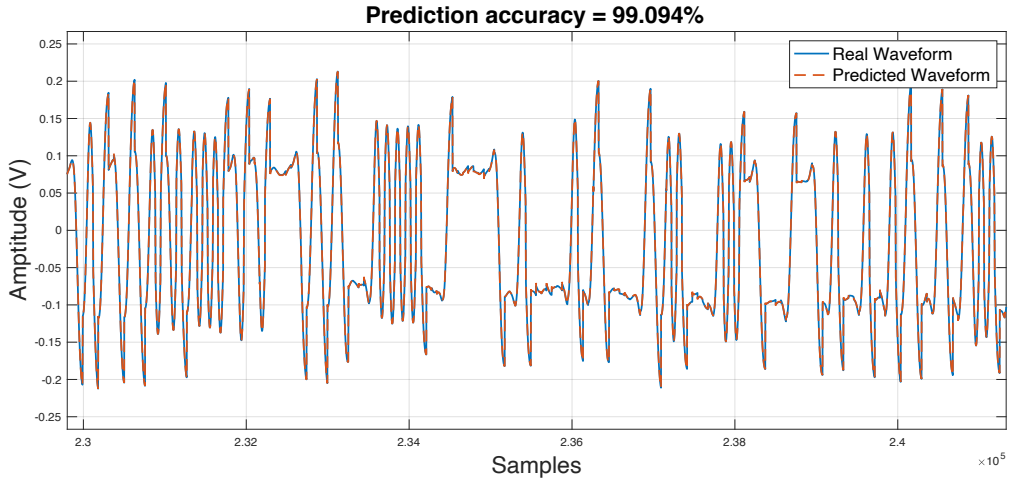
	Test case 1		Test case 2	
	RMSE	Worst Error	RMSE	Worst Error
Horizontal Bathtub	1.18e-13	2.80e-13	1.29e-12	2.08e-12
Vertical Bathtub	1.35e-03	2.06e-03	1.06e-03	1.67e-03

Next, let's move to the DFE mode. As shown in Table 4.3, under one particular CTLE and DFE setting, the signals at the receiver pass the eye margin test for three different TX settings and channels. In that case, the receiver setting is fixed during the measurement of the input and output data of the receiver. At first, the model is trained on the data from a high insertion loss channel and a medium TX swing. After the ANNARX model is well trained, the one-time-trained model will be tested under two test cases. The first test case is from a medium insertion loss channel and a low TX swing, and the other test case is from a high insertion loss channel and a high TX swing.

Two test case results are shown in Figure 4.14. The proposed ANNARX model shows high correlation with the circuit transient behavior. The worst prediction accuracies for both cases are 98.2% and 98.0%. The model can predict low and high frequency gain accurately. The prediction-generated eye diagram and bathtub curve are also compared with the real eye diagram and bathtub curve, which are shown in Figure 4.15 and Figure 4.16.

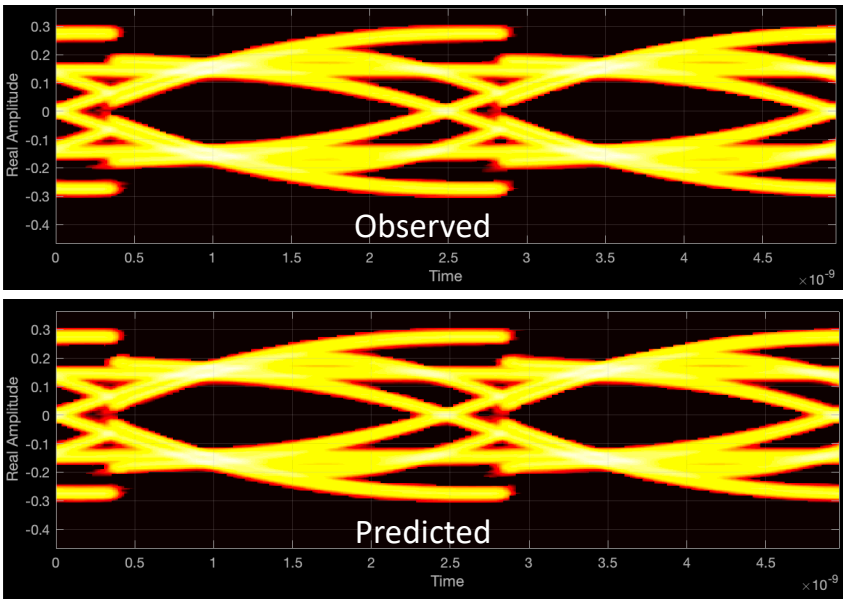


(a) Transient waveform prediction for test case 1

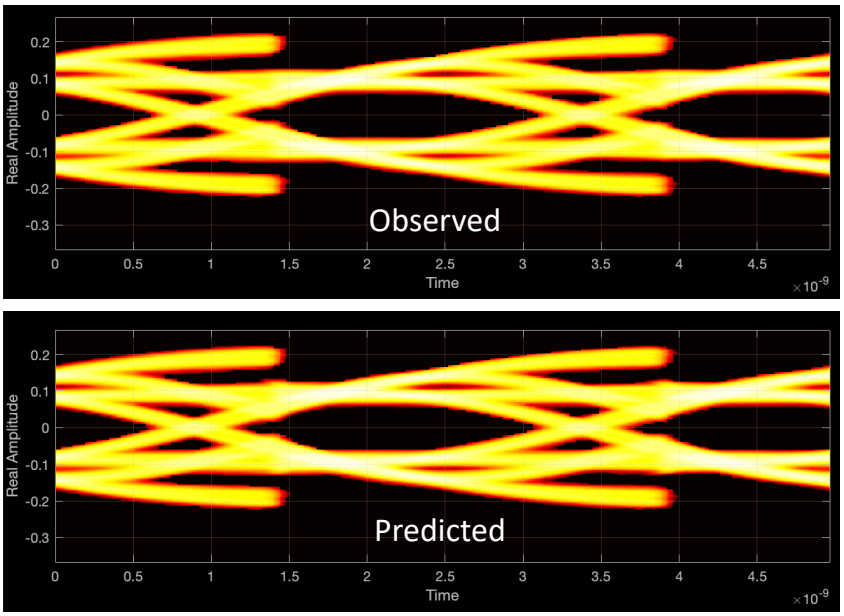


(b) Transient waveform prediction for test case 2

Figure 4.14. Waveform prediction results using the ANNARX model in DFE mode



(a) Eye diagram comparison for test case 1

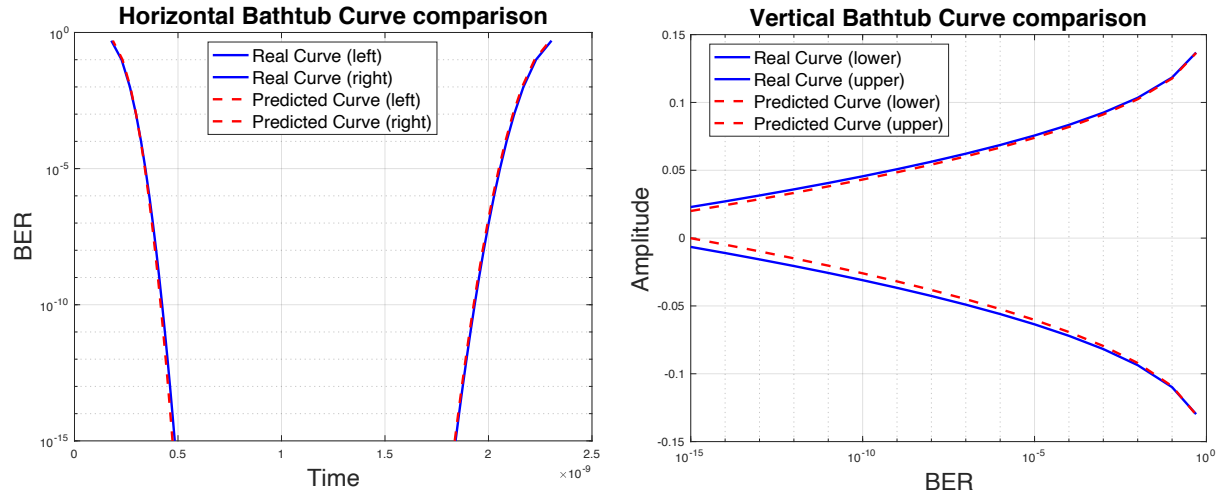


(b) Eye diagram comparison for test case 2

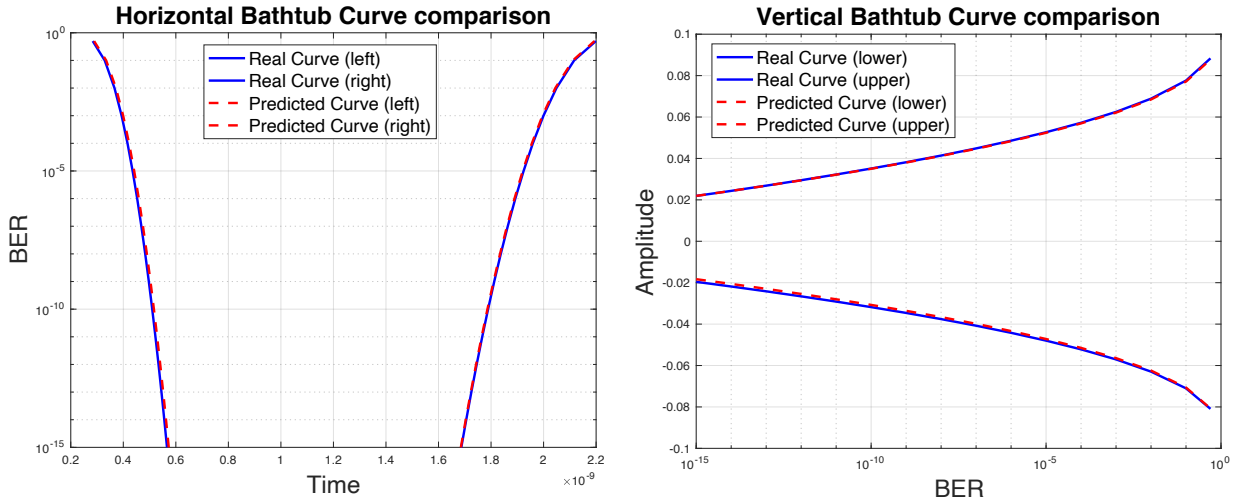
Figure 4.15. Real vs prediction eye diagram comparison in DFE mode

In Figure 4.15, the eye diagram generated from the predicted transient waveform has a high correlation with the real eye diagram. The inner eye and outer eye shapes are the same. Table 4.6 shows the eye height and eye width prediction results. The proposed ANNARX model can provide

high-precision eye margin prediction. The prediction results are better than other related works [9] [10] [11] [12].



(a) Bathtub curve comparison for test case 1



(b) Bathtub curve comparison for test case 2

Figure 4.16. Real vs prediction bathtub curve comparison in DFE mode

For the bathtub curve prediction in Figure 4.16, the bathtub curve generated from the predicted transient waveform has high correlation with the real curve horizontally and vertically. The RMSE and the worst error for different cases are shown in Table 4.7.

Table 4.6. Eye margin prediction results for DFE mode

	Test case 1			Test case 2		
	Real	Prediction	Accuracy	Real	Prediction	Accuracy
Eye height (V)	0.18	0.18	100%	0.12	0.12	100%
Eye width (ns)	1.34	1.32	98.5%	0.94	0.91	96.8%

Table 4.7. Bathtub curve prediction results for DFE mode

	Test case 1		Test case 2	
	RMSE	Worst Error	RMSE	Worst Error
Horizontal Bathtub	6.33e-12	6.79e-12	8.41e-12	9.21e-12
Vertical Bathtub	6.40e-03	9.35e-03	1.10e-03	1.28e-03

The input orders of the ANNARX models for the CTLE and DFE mode are 15 and 61 respectively, while the output orders of the ANNARX models for the CTLE and DFE mode are 32 and 78 respectively. From the experiments of Section 4.1.2 and 4.2.5, the order of the NNARX model would increase with the increasing number of DFE taps. However, the increment of the order is not linear. The reason is that the internal structures of NNARX model and DFE circuit are different. The current output of the DFE is determined by previous bit logic values (0 or 1). On the other hand, the output of the NNARX model is determined by the previous sampling points of the input and output data. Hence, it's better to use the proposed ANNARX model to self-determine the input and output order for each receiver setting.

To test the robustness of the proposed model, the cross-validation method is used for the CTLE and DFE mode modeling. Table 4.8 shows the time domain prediction accuracies for the cross-validation method. Case 1 and Case 2 in the table are training case and testing case 2 from

the previous experiment. Using different data case as the training source, the proposed model can also provide robust predictions.

Table 4.8. Cross-validation results for CTLE and DFE mode

	CTLE mode				**DFE mode**			
Training source	Case 1		Case 2		Case 1		Case 2	
	Average accuracy	Worst accuracy	Average accuracy	Worst accuracy	Average accuracy	Worst accuracy	Average accuracy	Worst accuracy
Testing case 1	99.93%	99.21%	99.94%	99.19%	99.13%	98.52%	99.11%	98.37%
Testing case 2	99.91%	99.15%	99.89%	99.09%	99.03%	98.49%	98.91%	98.01%

4.3. Single model for one receiver setting vs. one model for multiple receiver settings

The proposed model has a high correlation with the single receiver setting behavior. However, if all the receiver settings are taken into consideration, numerous models should be built to cover all the behaviors for different receiver settings, which takes a tremendously long time. The reason why the proposed model can only mimic one single receiver setting behavior is that the NNARX model is a single input and single output model, in which the input and output size of the model is one. Hence, only one receiver input and output data pair can be used to train an NNARX model. The LSTM model can handle multiple inputs. The receiver input waveform and receiver settings would be the inputs of the LSTM model, while the output is the predicted output waveform.

The measured data from Section 4.2.1 are used to train a multiple-input LSTM model for two receiver settings, one for CTLE mode setting, the other for DFE mode setting. Each receiver setting has three different input and output data under different channels and various TX setting cases. For each receiver setting, one input and output data are set as training data and the rest two

are set as testing data. The average overall accuracy and average worst accuracies for testing cases are measured. Table 4.9 shows the performance of the multiple-input LSTM model and the proposed ANNARX model in transient simulation. The proposed ANNARX model is much better than the LSTM model. The generated eye diagrams and bathtub curves using the LSTM-predicted waveforms also show low correlations with the real eyes and bathtub curves.

Table 4.9. Performance comparison between the LSTM and proposed ANNARX model

	LSTM		ANNARX	
	Average accuracy	Average worst accuracy	Average accuracy	Average worst accuracy
CTLE mode setting	95.53%	93.01%	99.93%	99.14%
DFE mode setting	94.62%	92.85%	99.90%	99.03%

For the simulation speed, the run time of the LSTM model is around 5 minutes for the half-million bits prediction, which is much slower than the ANNARX model.

Currently, for the receiver transient behavioral modeling, it's the best approach to build one model for one receiver setting. With the proposed model in this research, a receiver behavior library could be built, which contains various ANNARX models, to mimic the receiver equalization behavior.

4.4. Use case of the receiver transient behavioral model using machine learning

Using the proposed model, users can build receiver transient behavioral model for each receiver setting. The trained behavioral models can form a receiver behavioral library, which has two use cases.

The first use case is that engineers want to manually tune the receiver setting and do the signal integrity analysis. With the proposed model, engineers can quickly get the transient

waveform, eye diagram, and bathtub curve information under the current channel and current TX setting. The IBIS-AMI model can only do bit-by-bit simulation, which takes about 30 minutes for half-million-bit simulation, while the proposed behavioral model can predict half-million-bit waveform in 10 seconds. The simulation time using the proposed behavioral model is 180 times faster than the IBIS-AMI model. In that case, engineers can quickly do signal integrity analysis, for example, BER level analysis, using the simulated eye diagram and bathtub curve.

The second scenario is when engineers want to get a better receiver setting without doing the long-time transient simulation. Since the proposed model has a fast simulation speed, if a new channel or a new TX setting is changed, the behavioral model library can quickly predict all possible output waveforms and eye diagrams under different receiver settings and select a better receiver setting according to eye opening information, as shown in Figure 4.17. This is a new approach to do the receiver adaptation.

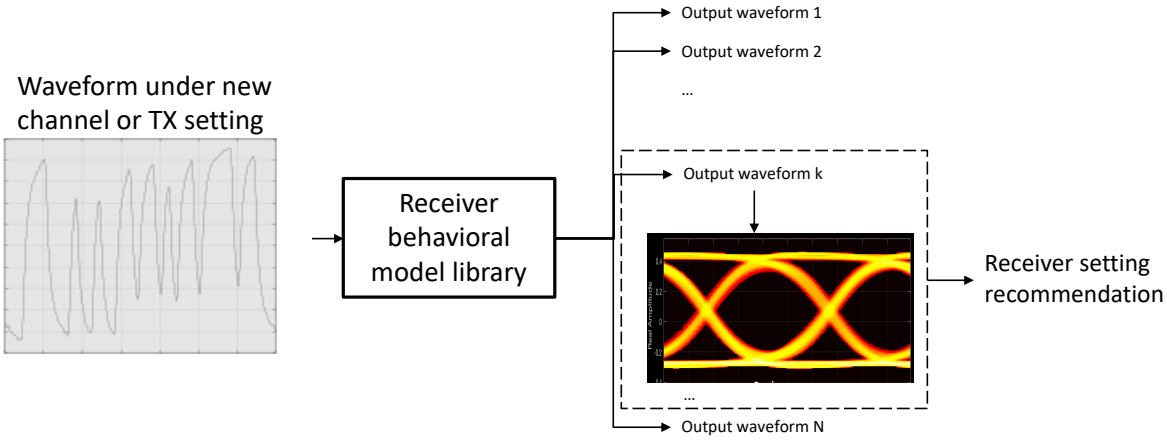


Figure 4.17. Receiver setting recommendation

However, there are lots of combinations of the receiver settings, which is impossible to build behavioral models for all the settings. The solution is that only low, medium, and high value of each receiver parameter are considered. According to Figure 4.17, users can get a receiver setting recommendation according to the simulated eye-opening. And users would know the

approximate range for each receiver setting, for example, CTLE low-frequency gain is low and DFE tap1 is high, under the current channel and TX setting, which could speed up the manually tuning time for the engineers. Traditional simulation methods, like IBIS-AMI model, can only do serial transient simulation, which would consume a very long time. With the receiver behavioral model library, because of the individuality of each receiver behavioral model, the transient waveforms for all the receiver settings can be simulated in parallel, which significantly speeds up the simulation time.

4.5. Conclusion

Since the IBIS-AMI model has a long simulation time and no current research can do bathtub curve prediction, an adaptive-ordered system identification model is proposed, which can do high-precision transient simulations. The model can provide transient waveform, eye diagram, and bathtub curve simulation at the same time. The transient waveforms prediction accuracies are above 99%. The eye height and eye width prediction accuracies are also above 96.5%. The proposed model can also provide a high-correlation eye shape and bathtub curve prediction. Moreover, the simulation speed of the proposed model is 10 seconds for a half-million-bit simulation, which is 180 times faster than the IBIS-AMI model. With our work, users can quickly get transient waveform, eye diagram and bathtub curve simulation results under a specific receiver setting.

Since the receiver behavior would change under different voltage or temperature, future work will focus on building receiver behavioral models across the process, voltage, and temperature (PVT) test. The initial solution is to add more training cases under the impact of PVT variations during the model training process.

Chapter 5. Receiver adaptation behavioral modeling

The receiver consists of a CTLE and a DFE, which are used to mitigate channel loss and ISI. The CTLE and DFE settings can work in two ways. One way is the fixed configuration, which means the gain of each stage is fixed. The problem of this approach is that it cannot suite for different cases. Hence the adaptive equalization system is popular in the industry. In the adaptive system, the gain at each stage is adaptive, which determined by the signal transmitting to the receiver. The goal of the adaptation is to automatically tune the CTLE and DFE settings to achieve larger eye-opening, or lower BER. Because the adaptation algorithm can adjust CTLE and DFE settings, the same transceiver design can build a robust serial link under different channels [41].

In this work, a Cascaded Deep Learning (CDL) modeling mechanism is proposed, which shows a parallel approach to modeling adaptive SerDes behavior effectively. The advantages of the proposed CDL model are illustrated in Figure 5.1. Specifically, the proposed modeling methodology provides fast and high-precision predictions under various PCB channels and different transmitter settings.

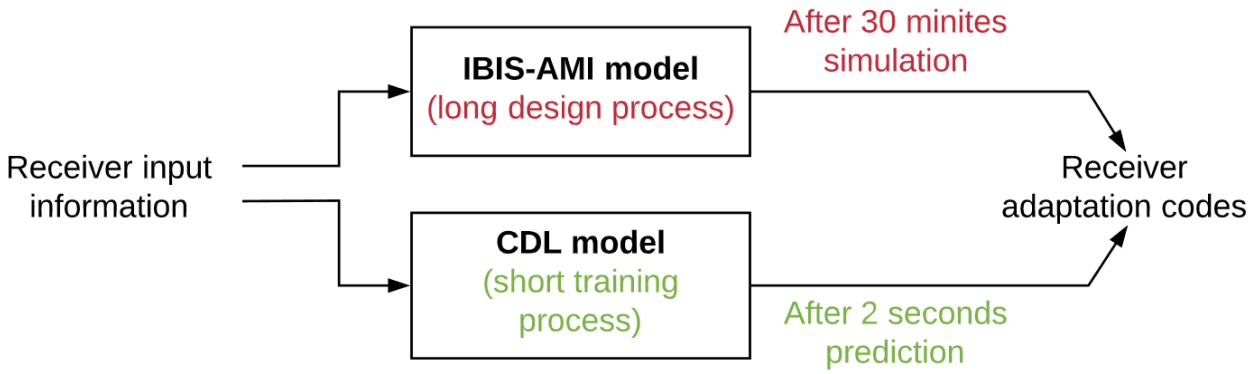


Figure 5.1. Traditional IBIS-AMI model vs CDL model in the receiver adaptation simulation

5.1. The challenges of the receiver adaptation modeling

The adaptation process allows the optimization of the equalizers for varying channels, environmental conditions, and data rates. One of the commonly used CTLE adaptation technique is CTLE tuning with output amplitude measurement [37]. As for the DFE adaptation, [39] proposed 2x oversampling the equalized signal at the edges can be used to extract information to adapt a DFE and drive a CDR loop, and sign-sign LMS algorithm used to adapt DFE tap values.

Because the adaptation process in the high-speed SerDes link is a non linear time-invariant (non LTI) system with many uncertainties. In a non LTI system, the system response inherently changes with the time for the same signal. Therefore, it's difficult to build a behavioral model.

5.1.1. Challenges in the industry

Currently, in the industry, only IBIS-AMI model has the capability to mimic the receiver adaptation process, which has a high correlation with the real circuit behavior. Chapter 3 describes how the IBIS-AMI models do the adaptation.

However, the problem of the IBIS-AMI model is that the simulation speed is slow. Normally, as for the adaptation simulation, engineers have to wait for a half-hour to see the adaptation results, because the adaptation process is based on the transient simulation. Furthermore, designing an IBIS-AMI model needs detailed information about the circuit design and lots of trial & errors to improve model performance. Normally, the silicon vendors would provide the IBIS-AMI model to the customers for them to do the test before they deliver the real products. It takes a long time, for example, one year, to build an IBIS-AMI model for one particular design, which affects their delivery time.

5.1.2. Challenges in machine learning

Nowadays, one way to skip a long design process and learn the circuit behavior is the machine learning approach. Machine learning models are widely used in SerDes applications. However, the adaptation process in the high-speed SerDes link is a non LTI system, which is difficult to build a behavioral model.

Currently, no research focuses on adaptation modeling using machine learning. Moreover, in the industry, the biggest challenge for building behavioral models is that a new model is needed to be designed for each product. And the model design process would cost lots of time and human efforts. In that case, a general machine learning modeling mechanism is required.

The goal of this work is to build a general modeling mechanism for the adaptation process in the high-speed receiver leveraging machine learning techniques. The model should provide high-precision adaptation codes in a short time during the simulation process.

5.2. Data generation

For the receiver adaptation system, the inputs are the waveform after the channel, while the outputs would be the receiver adaptation codes. In the real circuit, the adaptation can be done under all kinds of data patterns.

At first, a PRBS data pattern is sent to the receiver and after the adaptation process is done, the receiver codes are collected. The PRBS data pattern would be the model inputs, while the receiver codes are the model outputs. However, the prediction accuracies are very low. This is because the PRBS data contains redundant information, for example, data dependency, which are not related to the adaptation behavior. The adaptation is mainly based on the high-frequency and low-frequency contents after the channel. If the PRBS data are used, we need to increase the data

length to cover enough information for the adaptation process, which makes the training process tediously long.

From the previous experiment, efficient input data are needed to be fed to the machine learning model, which can help model learn the adaptation behavior. Therefore, a special data pattern is created, which contains a single bit response (SBR), and a long pulse response (LPR), shown in Figure 5.2. Because the adaptation algorithm is mainly based on the high-frequency and low-frequency content information after the channel, while SBR and LPR data can provide that information respectively. In the LPR, there are 100-bit zeros at the beginning, 150-bit ones in the middle, and 100-bit zeros at the end. For the SBR data, there are 100-bit zeros at the beginning, 1-bit one in the middle, and 100-bit zeros at the end. The length of the whole data pattern is 551 bits. For each data measurement, a special data pattern after the channel is sent to the receiver. After the adaptation process is finished, the special data pattern waveform after the channel and the adaptation codes, such as CTLE and DFE settings, are measured as the model inputs and outputs. The sampling rate for the receiver input is 1649.92 GHz (64 sample points per UI).

This special data pattern is sent to the receiver to do the adaptation. After the adaptation process is done, we collect this special data pattern waveform after the channel and the adaptation codes, such as CTLE and DFE settings. Hence, the special data pattern waveform after the channel can provide useful information for the adaptation behavioral modeling.

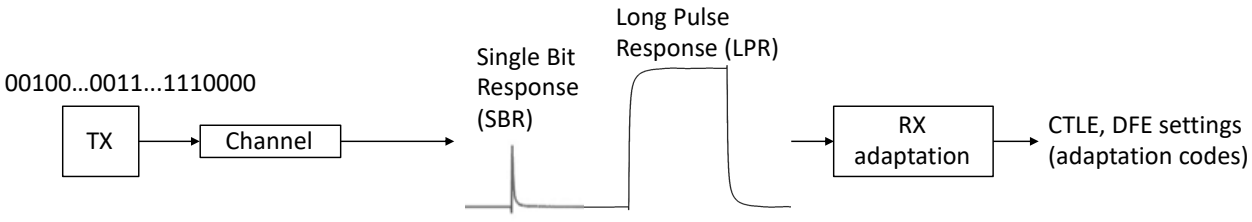


Figure 5.2. Data collection process

To prove the robustness of our modeling mechanism, the data are collected from two different designs, namely the Xilinx UltraScale+ GTY transceiver and Xilinx UltraScale+ GTH transceiver [41] [42]. The UltraScale+ GTY transceiver is a 16 nm device, supporting line rates from 500Mb/s to 32.75Gb/s. It provides maximum performance for the 28G backplane applications. In the GTY receiver, there are 3 CTLE stages and 15 DFE taps. The UltraScale+ GTH transceiver, which is a 20 nm device, supports line rates from 500Mb/s to 16.375Gb/s. The GTH receiver has 3 CTLE stages and 11 DFE taps.

For the UltraScale+ GTY and GTH receiver, the CTLE has 3 stages, namely high-frequency gain (KH), low-frequency gain (KL), and auto gain control (AGC), illustrated in Figure 5.3 [41]. The KL and KH are to boost low-frequency and high-frequency contents respectively after the channel. The AGC is to boost both low-frequency and high-frequency contents.

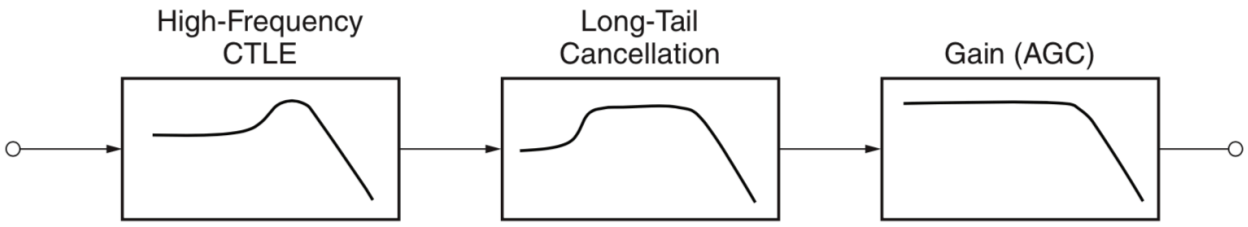


Figure 5.3. Three-stage CTLE in UltraScale+ GTH and GTY transceiver

In the UltraScale+ GTY and GTH transceiver, there are 15 DFE taps and 11 DFE taps respectively. Because GTY transceiver can cover a wider range of the line rate, more DFE taps are needed.

In Figure 5.2, if we tune the TX setting or switch the channel, the special waveform would change. The adaptation codes change accordingly. Thus, for each design, the data are measured under different channel cases and various TX settings. The data configurations for both designs are shown in Table 5.1. Those data are collected over various industry channels (FCI backplane). By tuning the TX settings, the adaptation codes are changed accordingly, and we can generate

several cases over one channel. Among all the TX settings, low, medium and high swing settings are selected because they can cover all the under equalized, equalized, and over equalized cases. The data from the UltraScale+ GTY transceiver would be used in the modeling mechanism exploration. The data from the Xilinx UltraScale+ GTH transceiver would be used in the robustness test of the proposed model.

Table 5.1. Data configuration for two transceiver designs

	Configurations
Data rate	25.78Gpbs
Channel insertion loss	Low, medium, high
TX settings	Low, medium, high swing

For the modeling mechanism exploration, 600 cases are measured from the UltraScale+ GTY transceiver. Since the adaptation results would be changed if the TX setting or the channel is changed, those 600 cases are generated by using different TX settings and different channels. There are 30 different channels, which have low, medium, or high losses. And the TX setting is swept from low to high. 450 data from 20 channels are set as the training dataset. 50 data from 4 channels are set as the validation data. The rest 100 data from 6 channels are set as testing data. Those 6 testing channels are hidden from the model training process. For the robustness test of the modeling mechanism, 200 cases are measured from the UltraScale+ GTH transceiver. 150 data from 6 channels are set as the training dataset. 20 data from 2 channels are set as the validation data. The rest 30 data from 3 channels are set as testing data. The model inputs are measured from the receiver input waveforms, which are SBR and LPR data patterns. The targets are collected from the receiver adaptation codes after the adaptation process is done.

5.3. Modeling mechanism exploration

5.3.1. Performance metrics

As for the data generation, receiver adaptation codes are measured after the adaptation process is done. However, because adaptation is a non-LTI system, even after the adaptation is done, the receiver equalization codes are still changing from bit to bit. Since the machine learning model target should be a single value, during the model training, the mean code values after the adaptation are used as the model targets, as shown in Figure 5.4. The mean square error (MSE) is used as the cost function during the training process, which can be obtained as

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{5.1}$$

where i is the index of data. y_i is the real adaptation code value. $\hat{y}_i$ is the model predicted value. m is the number of data.

During the model prediction, a target range, which can be calculated as $[mean - threshold, mean + threshold]$ is set for each receiver code. The mean is the mean value of each code after adaptation process. The threshold is set according to the range of each code after the adaptation. Hence, it's a correct prediction if the predicted value lies in the target range. The prediction accuracy can be calculated as:

$$Prediction\ accuracy = \frac{n}{m} \tag{5.2}$$

where n is the number of cases that the predicted value is in the target range. m is the number of all the testing cases. At the model testing process, the prediction accuracy is calculated, which can describe the model performance.

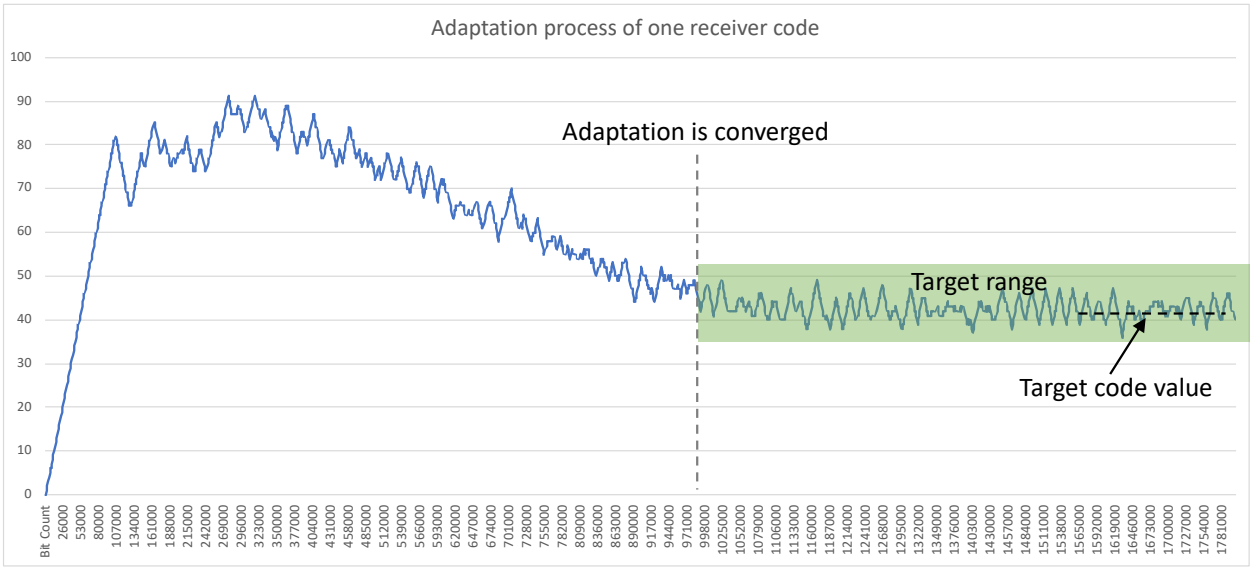


Figure 5.4. Adaptation process for one receiver code

Another performance metrics, used in the adaptation modeling, is called worst deviation. The worst deviation means the largest distance between the target range and the predicted value. It can describe the worst performance of the model.

5.3.2. Adaptation code prediction using LSTM model

After the data collection, we have the SBR and LPR time-series data as the model input information and the adaptation codes as the model targets. Because the LSTM model can extract useful information in the time-series data, a LSTM model is built to predict all the receiver adaptation codes using SBR + LPR data. The modeling chart is illustrated in Figure 5.5. Initially, the random search method is used to find the best model configuration for LSTM model.

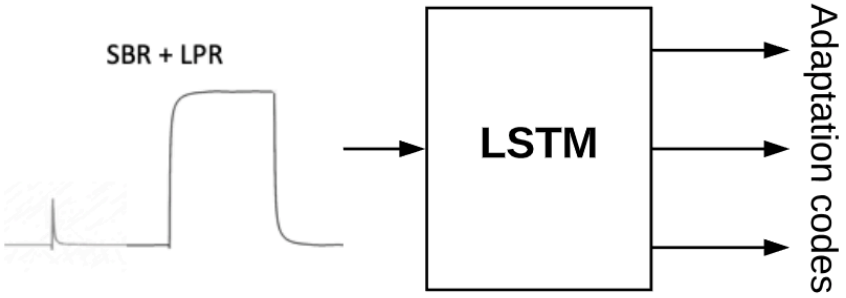


Figure 5.5. Adaptation code prediction using the LSTM model

After training, the LSTM prediction results using the SBR + LPR data are shown in Table 5.2. The prediction accuracies for most of DFE taps, which highlighted in yellow, are above 70%, while CTLE codes and some DFE taps show low correlation with the real targets. The overall model performance needs be improved.

Table 5.2. Prediction results using one LSTM model

Targets	Accuracy	Worst deviation	Targets	Accuracy	Worst deviation
CTLE AGC	12%	8	DFE Tap7	72%	7
CTLE KL	7%	9	DFE Tap8	68%	9
CTLE KH	9%	9	DFE Tap9	79%	5
DFE Tap1	76%	19	DFE Tap10	79%	6
DFE Tap2	68%	18	DFE Tap11	82%	5
DFE Tap3	66%	17	DFE Tap12	81%	4
DFE Tap4	73%	9	DFE Tap13	88%	3
DFE Tap5	82%	10	DFE Tap14	85%	2
DFE Tap6	71%	7	DFE Tap15	83%	2

5.3.2. CTLE and DFE basics in modeling flow design

To improve the model performance, some basic knowledges of CTLE and DFE should be considered during the modeling flow design.

As introduced in Section 2.1, the CTLE would attenuate the low frequency content and boost high frequency content. Typically, CTLE settings have strong relationship with high-frequency and low-frequency information, which can be measured from the SBR + LPR data pattern. The peak value of the SBR can represent high-frequency loss after the channel, while the average value at the DC level of the LPR can represent low-frequency gain. The difference between high-frequency and low-frequency content represents the channel loss. Therefore, low-frequency and high-frequency signal amplitude are measured from SBR and LPR data pattern,

shown in Figure 5.6. The two measured features, namely low-frequency and high-frequency signal amplitude, can be used to predict the CTLE codes.

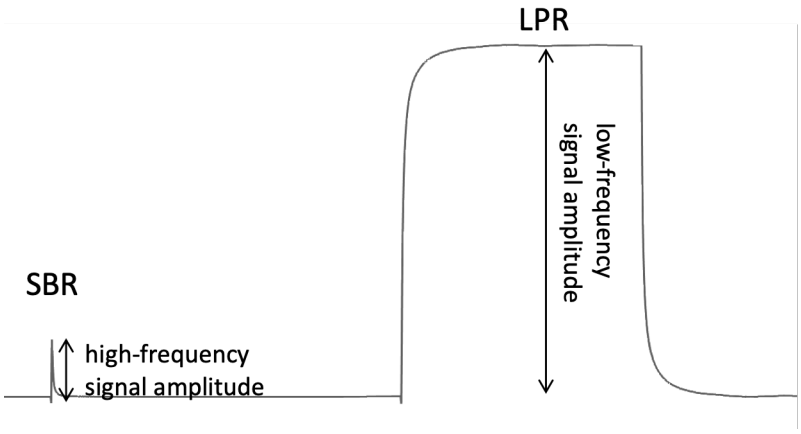


Figure 5.6. Low-frequency and high-frequency signal amplitude measurement

As for the DFE, in the real circuit, the DFE tap values are calculated by the postcursor information after the SBR main cursor. Consequently, each DFE tap value has a very strong relation with the SBR postcursor information. The DFE postcursor cancellation capabilities are determined by the filter length, as shown in Figure 5.7. Accordingly, SBR information would be considered to predict DFE taps.

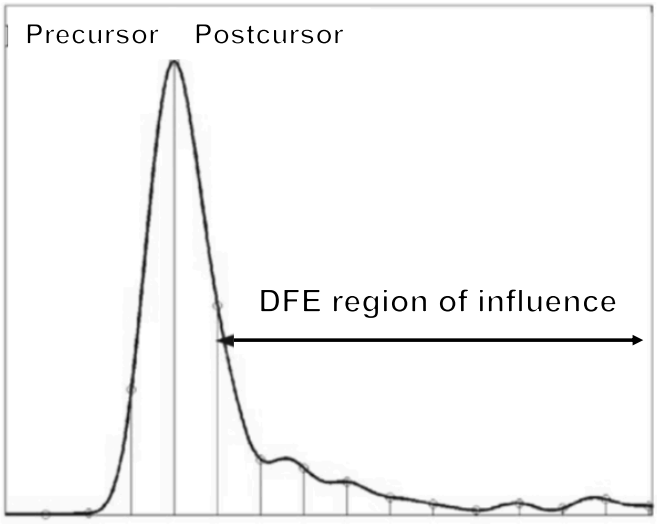


Figure 5.7. DFE cancellation region

5.3.3. Adaptation code prediction using DNN + LSTM model

It would be better to use the two measured features from SBR + LPR data pattern, namely low-frequency and high-frequency signal amplitude, to predict the CTLE codes and SBR data pattern to predict DFE taps.

For the CTLE prediction, multiple model types, for example, SVM, Decision Tree, and DNN, are tried. The DNN model shows the best performance. Consequently, the DNN model type is used for the CTLE adaptation code prediction. For DFE taps, only postcursor data from the SBR are sliced according to UI information and used to predict the DFE taps. Each DFE tap is predicted using one LSTM model. The input of the LSTM model is the sliced SBR data, which is illustrated in Figure 5.8. For example, the first UI after the main cursor would be sent to a LSTM model to predict DFE tap1 and the second UI after the main cursor is sent to another LSTM model to predict DFE tap2.

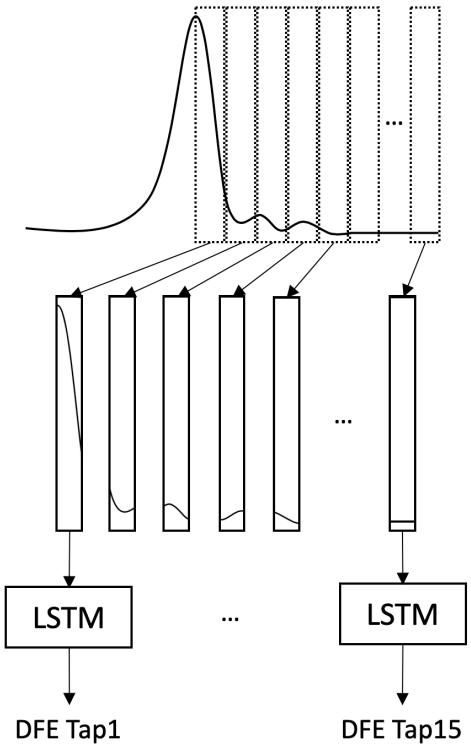


Figure 5.8. Sliced SBR data as the input of LSTM model

The DNN + LSTM model structure is used to predict the adaptation codes. The modeling flow is shown in Figure 5.9. Two features, namely low-frequency and high-frequency signal amplitude measured from the SBR and LPR data pattern, are used to predict 3 CTLE adaptation codes using one DNN model, while the sliced SBR data are used to predict DFE taps using individual LSTM models.

During each model training process in Figure 5.9, Bayesian optimization (BayesOpt) [44] [46] is applied to find the best hyperparameters for the model, which would improve the model performance using the minimum search. BayesOpt can optimize the performance of the machine learning model [44] [45].

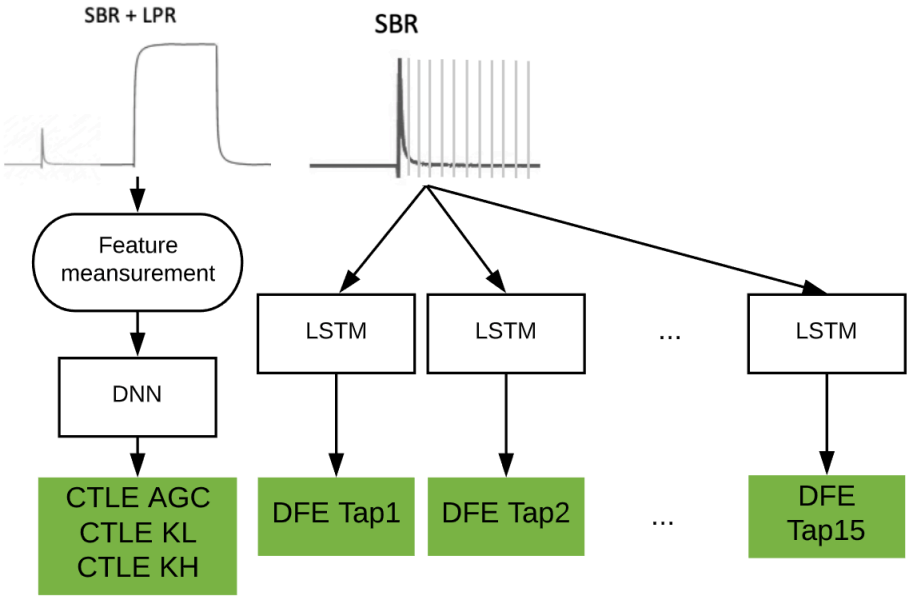


Figure 5.9. DNN + LSTM modeling mechanism

Generally, the goal of BayesOpt is to find a global maximum/minimum of an unknown objective function f

$$\theta^* = \underset{\theta \in \mathcal{X}}{argmin}\, f(\theta) \tag{5.3}$$

where $\mathcal{X} \subseteq \mathbb{R}^{D_X}$ is the design space or data space. $D_{\mathcal{X}}$ is the dimensionality of the parameter space. Furthermore, it is assumed that the unknown objective function f has can be evaluated at any arbitrary query point θ in the data space. This evaluation produces outputs $y \in \mathbb{R}$ such that $\mathbb{E}[\mathrm{y}|f(\theta)] = f(\theta)$. Hence, we can only observe the function f through unbiased noisy point-wise observations y. In this setting, we consider a sequential search algorithm which, at iteration n, selects a location θ_{n+1} at which to query function f and observe y_{n+1}. After N queries, the algorithm makes a final recommendation θ_N, which is the best estimate. BayesOpt proposes a prior belief over the possible objective functions and improves this model according to data observed via Bayesian posterior updating. Equipped with this probabilistic model, BayesOpt can sequentially induce acquisition functions Υ_n: $\mathcal{X} \longmapsto \mathbb{R}$ that use the uncertainty in the posterior to guide the exploration. Intuitively, the acquisition function evaluates the candidate points for the next evaluation of f; therefore, θ_{n+1} is selected by maximizing Υ_n, where the index n indicates the implicit dependence on the currently available data.

After modeling, the DNN + LSTM model can provide better performance than a single LSTM model. The prediction results are shown in Table 5.3. The CTLE code and DFE tap 1 – 3 predictions are much improved. From this experiment, the sliced data after the main cursor of SBR can be proved to predict DFE tap 1 – 3 using three individual LSTM models, while other predicted DFE taps show low correlations. This is because the long tail cancelation is mainly based on the first three DFE tap values. In the SBR, the signal after 3 UIs from the main cursor would become flat and contain few useful information. Hence, it's not practical to predict DFE tap 4 – 15 using the sliced SBR data. A different modeling flow is needed for the DFE tap 4 – 15 prediction.

Table 5.3. Prediction results using DNN + LSTM model

Targets	Accuracy	Worst deviation	Targets	Accuracy	Worst deviation
CTLE AGC	97%	1	DFE Tap7	73%	5
CTLE KL	92%	2	DFE Tap8	74%	6
CTLE KH	93%	2	DFE Tap9	79%	4
DFE Tap1	95%	4	DFE Tap10	79%	4
DFE Tap2	92%	3	DFE Tap11	82%	3
DFE Tap3	98%	1	DFE Tap12	81%	2
DFE Tap4	87%	6	DFE Tap13	82%	3
DFE Tap5	85%	6	DFE Tap14	84%	2
DFE Tap6	73%	7	DFE Tap15	83%	2

5.4. Cascaded Deep Learning modeling mechanism

In this section, the Cascaded Deep Learning (CDL) modeling mechanism is introduced. The cascaded model structure is often used in computer vision problems [49] [50]. The deep cascade of convolutional neural networks (CNN) is used to improve model performance. Currently, no research applies the cascaded model structure in the SerDes simulation area.

5.4.1. Modeling structure

As mentioned in Section 5.3, with high-frequency and low-frequency signal amplitude information, the DNN model is able to predict three CTLE adaptation codes. The sliced data after the main cursor of SBR can be proved to predict DFE tap 1 – 3 using three individual LSTM models. On the other hand, the DFE tap 4 – 15 show low correlations with the real tap values.

In the receiver, the DFE tap values are influenced by the previous DFE taps, because of the correlations among the DFE shift registers. Hence, the previous DFE taps have high correlation with the following DFE taps. Meanwhile, CTLE adaptation codes may also influence the DFE tap

values, because there are interactions among the CTLE and DFE settings during the adaptation process.

In that case, we propose a modeling mechanism, named Cascaded Deep Learning (CDL), to deal with general black-box problems in SerDes link. The proposed modeling architecture is shown in Figure 5.10.

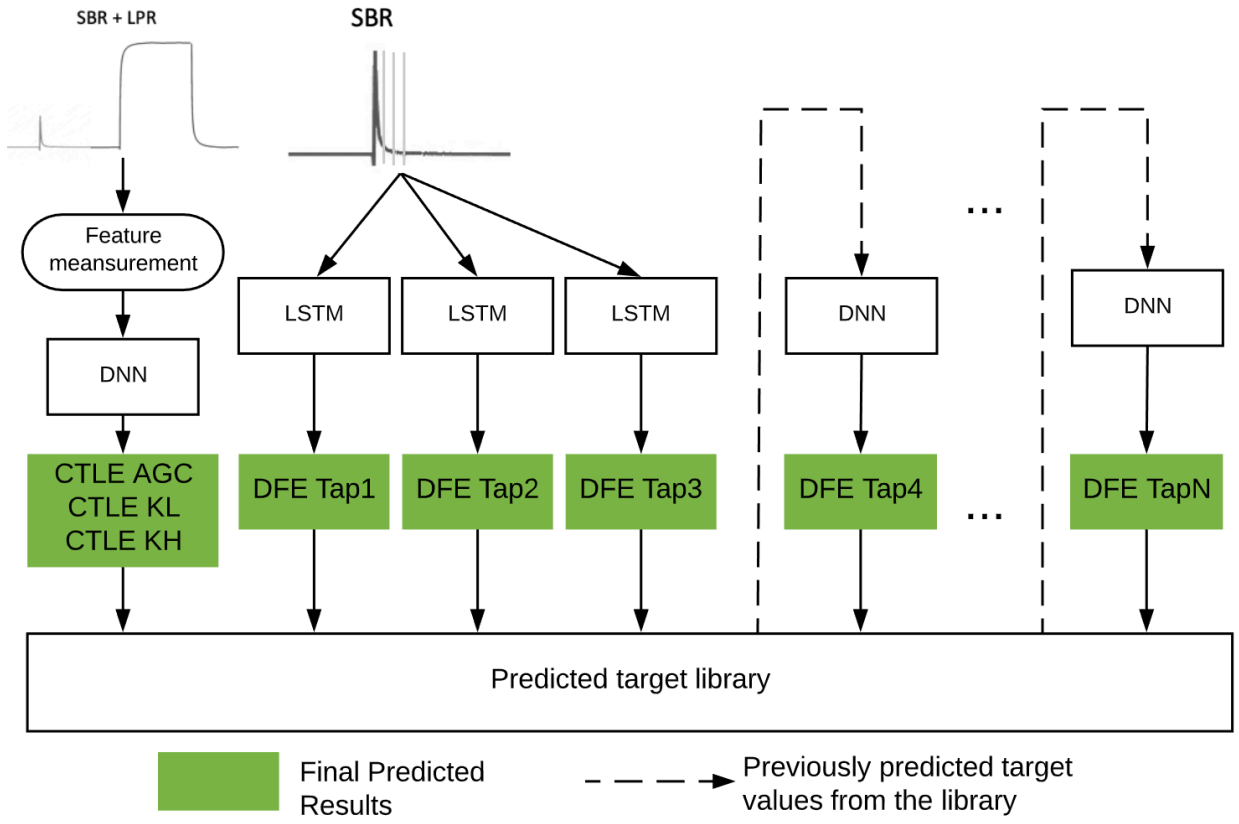


Figure 5.10. Cascaded Deep Learning modeling mechanism

As for the CTLE, a DNN model is used to predict the three CTLE adaptation codes. The inputs of the DNN model are two measured features from the SBR and LPR data pattern. As for the DFE tap1 – 3, one LSTM model is used for each DFE tap prediction. The input of the LSTM model is the sliced SBR information. Three predicted CTLE codes and three predicted DFE tap values will be saved in a predicted target library (PTL). Since the DFE tap values have high

correlations with previous DFE taps and CTLE codes, all the previous predicted values are sent to a DNN model for the current DFE prediction.

Table 5.4. Best model configurations for the proposed modeling mechanism

Targets	Model type	Configurations	Targets	Model type	Configurations
CTLE AGC CTLE KL CTLE KH	DNN	Neurons in each layer: (9, 35, 20, 41, 19, 7, 5); learning rate: 0.015	DFE Tap7	DNN	Neurons in each layer: (69); learning rate: 0.01
			DFE Tap8	DNN	Neurons in each layer: (57); learning rate: 0.01
			DFE Tap9	DNN	Neurons in each layer: (45); learning rate: 0.01
DFE Tap1	LSTM	Hidden units in each layer: (45, 45); learning rate: 0.021	DFE Tap10	DNN	Neurons in each layer: (51); learning rate: 0.01
DFE Tap2	LSTM	Hidden units in each layer: (52, 52); learning rate: 0.020	DFE Tap11	DNN	Neurons in each layer: (29); learning rate: 0.01
DFE Tap3	LSTM	Hidden units in each layer: (49, 49); learning rate: 0.018	DFE Tap12	DNN	Neurons in each layer: (28); learning rate: 0.01
DFE Tap4	DNN	Neurons in each layer: (78); learning rate: 0.01	DFE Tap13	DNN	Neurons in each layer: (19); learning rate: 0.01
DFE Tap5	DNN	Neurons in each layer: (74); learning rate: 0.01	DFE Tap14	DNN	Neurons in each layer: (10); learning rate: 0.01
DFE Tap6	DNN	Neurons in each layer: (65); learning rate: 0.01	DFE Tap15	DNN	Neurons in each layer: (11); learning rate: 0.01

After the current DNN model finishes its training, the current predicted DFE tap values will be saved in the PTL and used for the next DFE tap predictions until the last prediction target is reached.

For DFE tap 4 - 15, the previous predicted codes are used. In that case, cascaded deep learning models are structured to predict all the receiver adaptation codes. The cascaded deep learning model shows better performance than DNN + LSTM model.

After the model training process, the BayesOpt method finds the best configurations for each model in the proposed modeling mechanism, which are shown in Table 5.4.

5.4.2. CDL model prediction results

In this section, the prediction results of the CDL model are presented. 100 testing data are collected under 6 testing channels and various TX settings in the UltraScale+ GHY transceiver. Those testing data are hidden during the model training process.

After the CDL model cascaded training process, the prediction results for the CTLE adaptation codes and some of the DFE taps are shown in Figure 5.11, and

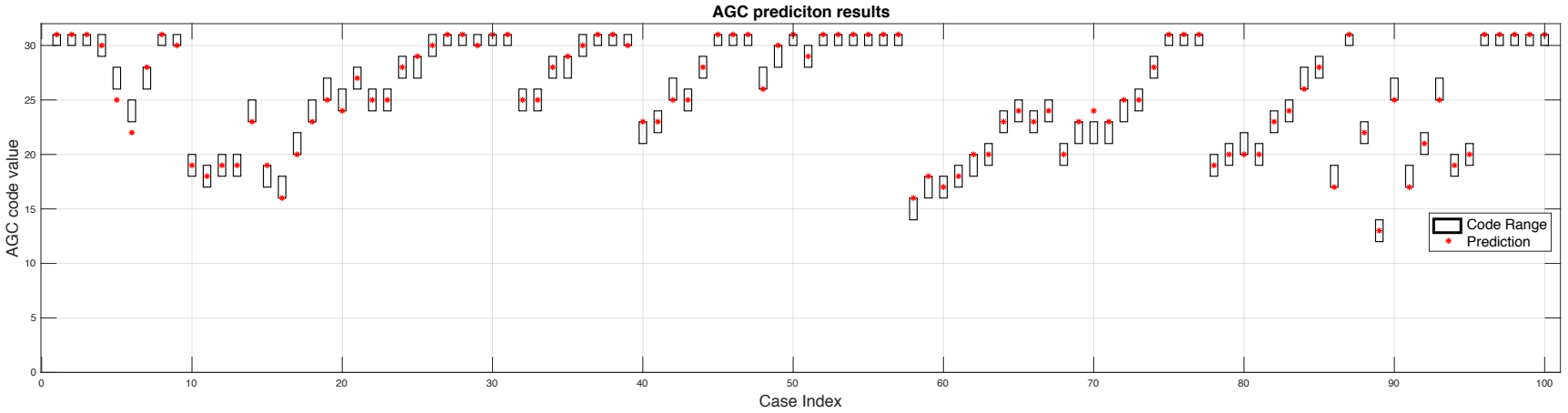
Figure 5.12 respectively. Because each code is varying during the adaptation process, a black box is used for each case, which represents a target range. The red stars are the CDL model prediction values. X axis is the case index number. Y axis is the code value. From the results, most of the predicted code values lie in the boxes. The predicted adaptation codes show high correlations with the real codes. Other DFE tap prediction results are shown in Appendix B.

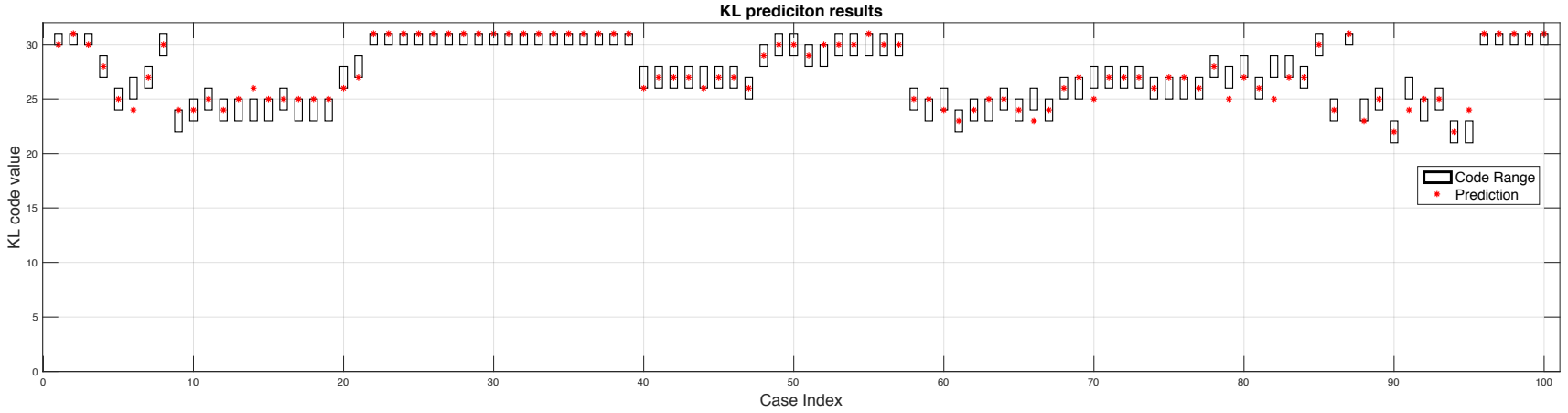
The prediction accuracies and worst deviations for all the adaptation codes are shown in Table 5.5. The prediction values have high correlations with the adaptation codes in the circuit. The worst performance of each model is also reliable. The simulation speed of each testing case is 2 seconds on average.

Table 5.5. Prediction results using the CDL model

Targets	Accuracy	Worst deviation	Targets	Accuracy	Worst deviation
CTLE AGC	97%	1	DFE Tap7	98%	2
CTLE KL	92%	2	DFE Tap8	98%	2
CTLE KH	93%	2	DFE Tap9	96%	2
DFE Tap1	95%	4	DFE Tap10	98%	2
DFE Tap2	92%	3	DFE Tap11	98%	1
DFE Tap3	98%	1	DFE Tap12	100%	0
DFE Tap4	95%	1	DFE Tap13	98%	1
DFE Tap5	92%	2	DFE Tap14	100%	0
DFE Tap6	95%	2	DFE Tap15	100%	0

Figure 5.11. CTLE code (AGC, KL, and KH) prediction results in the UltraScale+ GTY


AGC prediciton results
AGC code value
Case Index
Code Range
Prediction


KL prediciton results
KL code value
Case Index
Code Range
Prediction

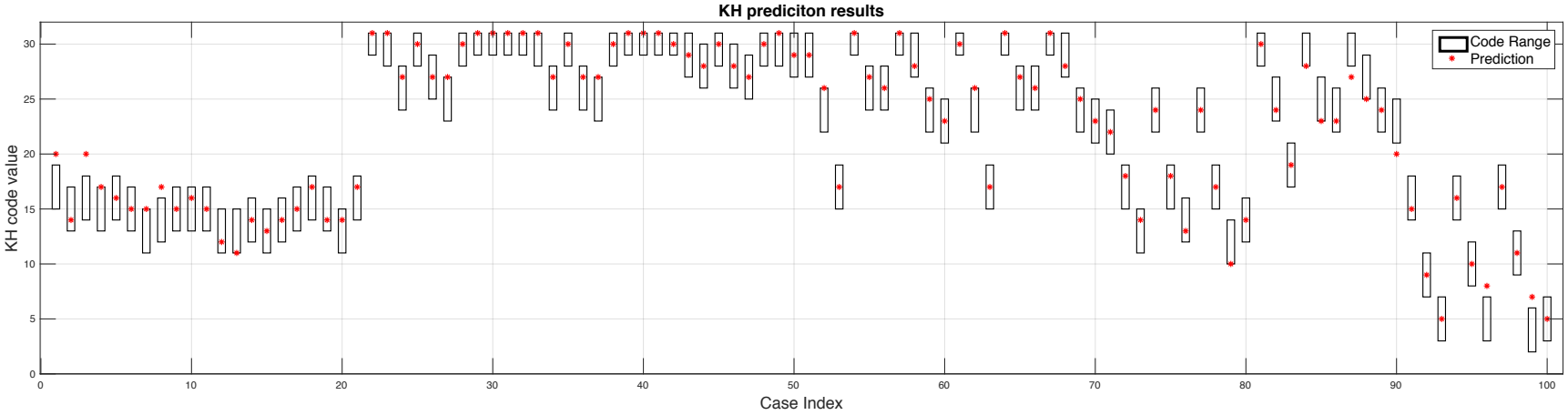
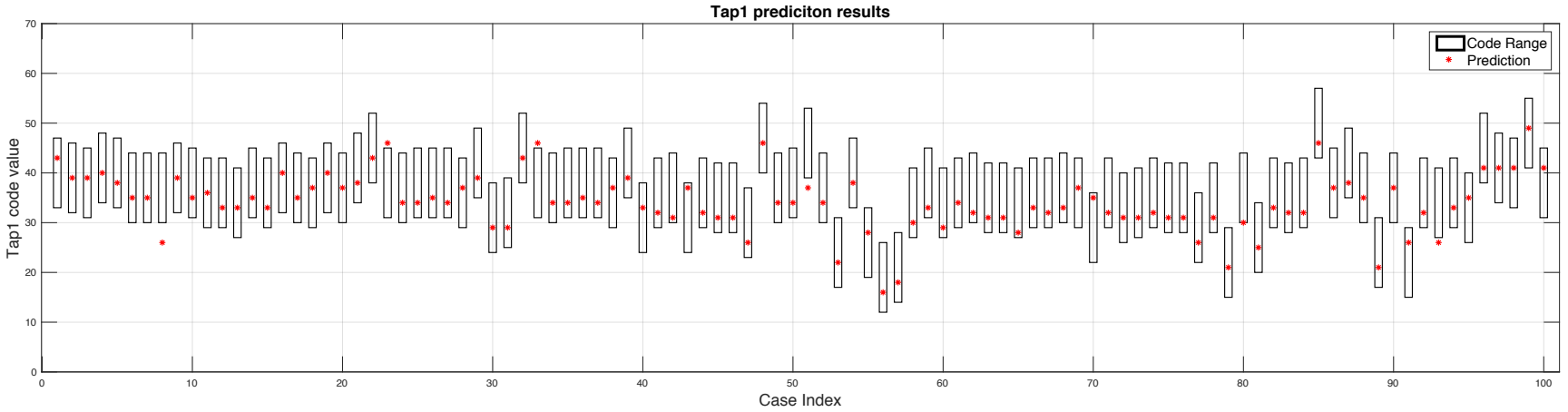
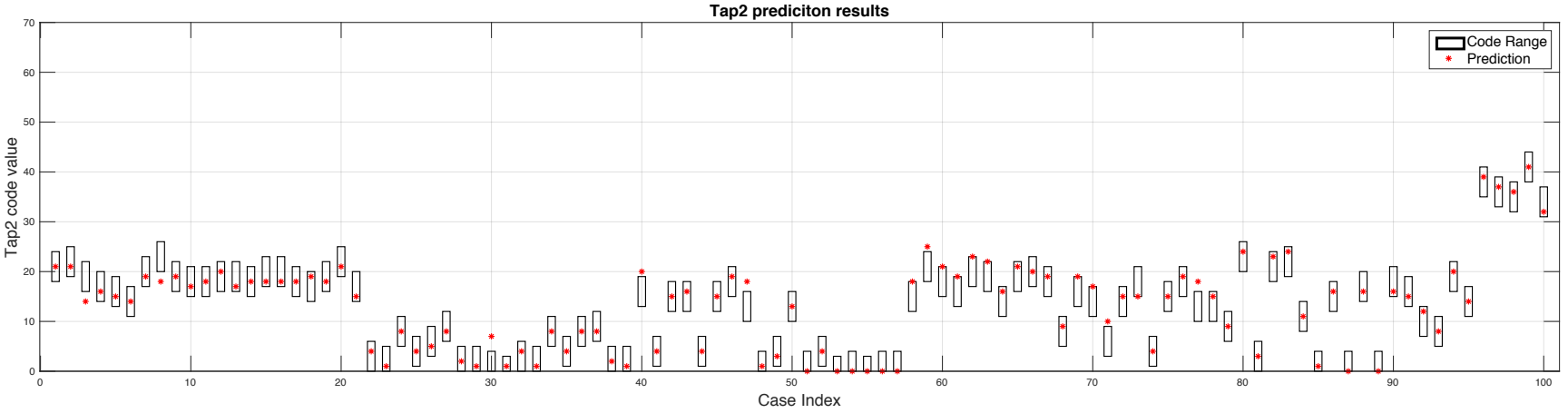
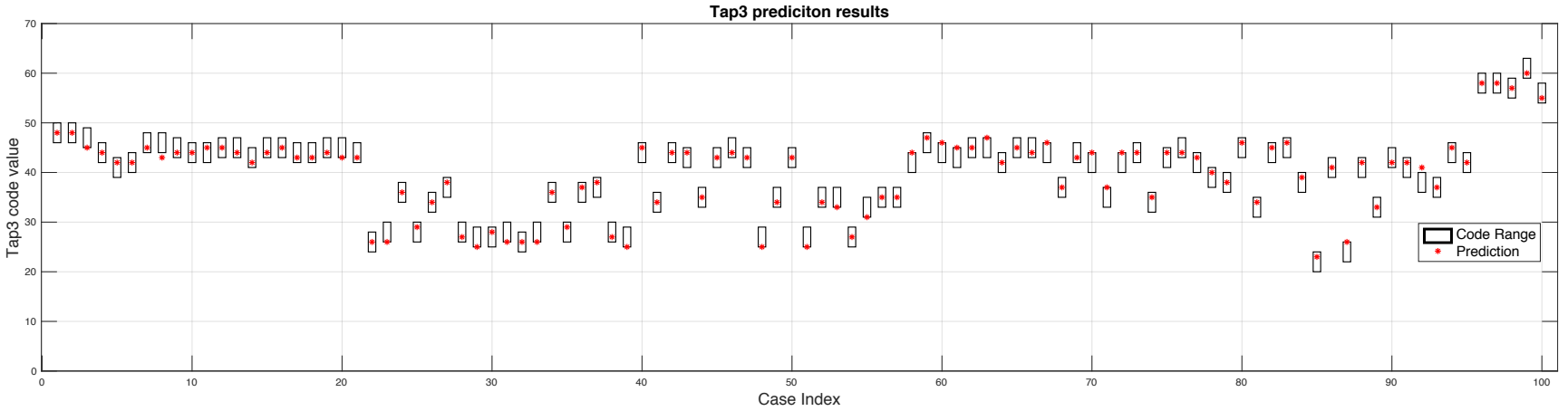
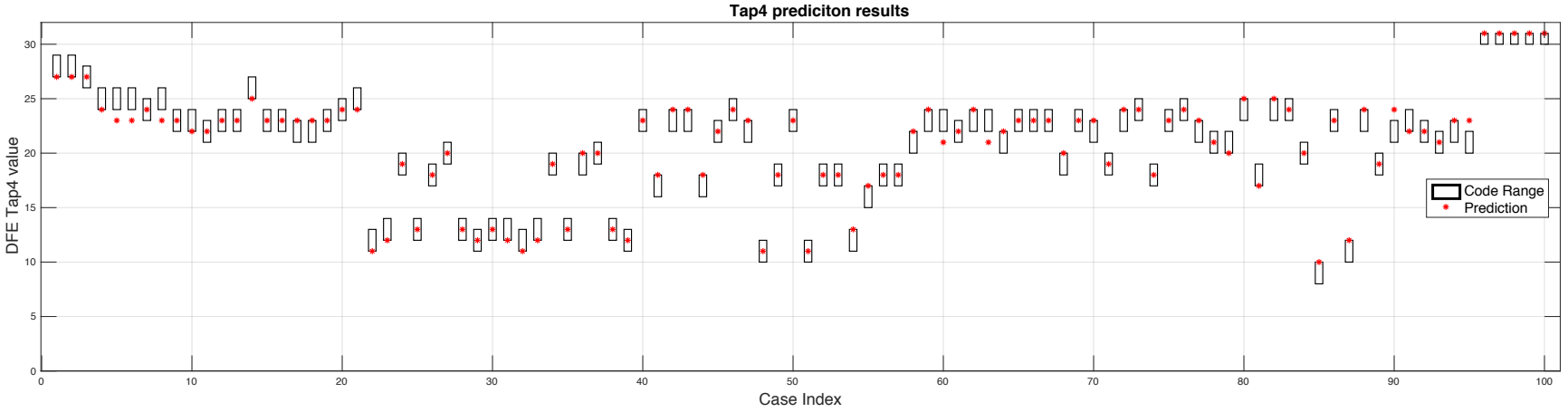

KH prediciton results
Code Range
Prediction
KH code value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100

Figure 5.12. DFE Tap (DFE Tap1 - 4) prediction results in the UltraScale+ GTY


Tap1 prediciton results
Code Range
Prediction
Tap1 code value
Case Index
0
10
20
30
40
50
60
70
80
90
100


Tap2 prediciton results
Code Range
Prediction
Tap2 code value
Case Index
0
10
20
30
40
50
60
70
80
90
100


Tap3 prediciton results
Tap3 code value
Case Index
Code Range
Prediction


Tap4 prediciton results
DFE Tap4 value
Case Index
Code Range
Prediction

5.5. Robustness test

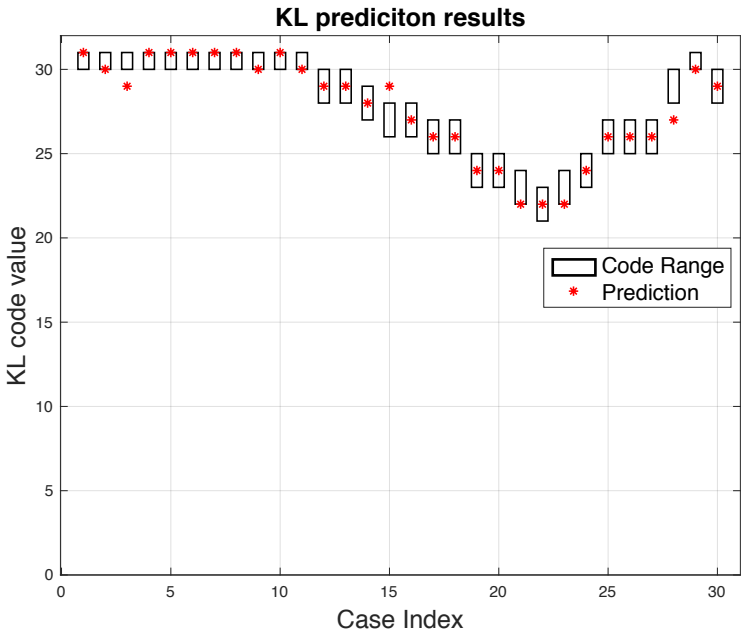
To test the robustness of the proposed CDL modeling mechanism, a different design, which is the UltraScale+ GTH transceiver [41] [42], is used. The UltraScale+ GTH receiver has 3 CTLE adaptation codes, namely AGC, KL and KH, and 11 DFE taps.

200 cases are measured from the UltraScale+ GTH transceiver. 150 data from 6 channels are set as the training dataset. 20 data from 2 channels are set as the validation data. The rest 30 data from 3 channels are set as testing data.

Using the CDL modeling mechanism to learn the adaptation behavior in the UltraScale+ GTH receiver, the CTLE and DFE testing results are shown in Figure 5.13 and Figure 5.14 respectively. The model prediction results show high correlation with the receiver adaptation codes.

Figure 5.13. CTLE adaptation code (AGC, KL, KH) prediction in UltraScale+ GTH receiver


AGC prediciton results
AGC code value
Case Index
Code Range
Prediction
0
5
10
15
20
25
30


KL prediciton results
KL code value
Case Index
Code Range
Prediction
0
5
10
15
20
25
30

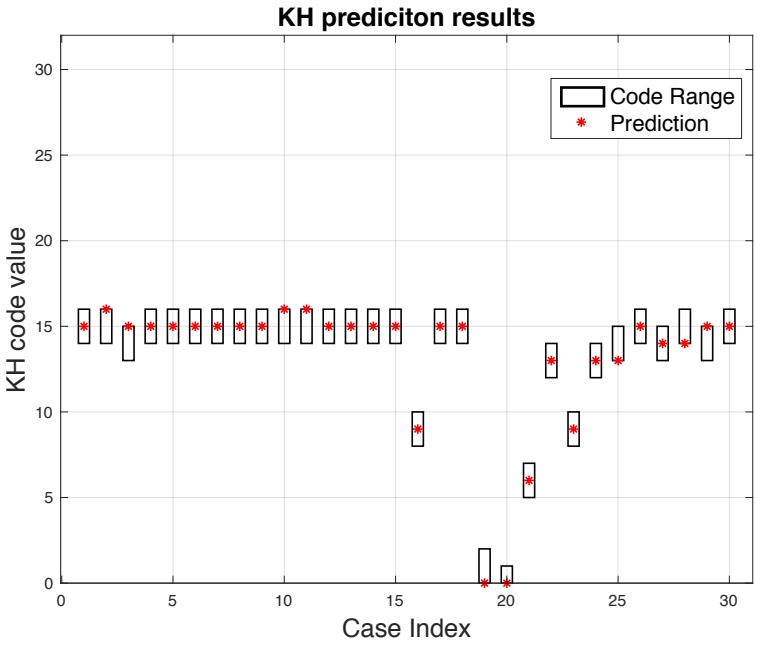
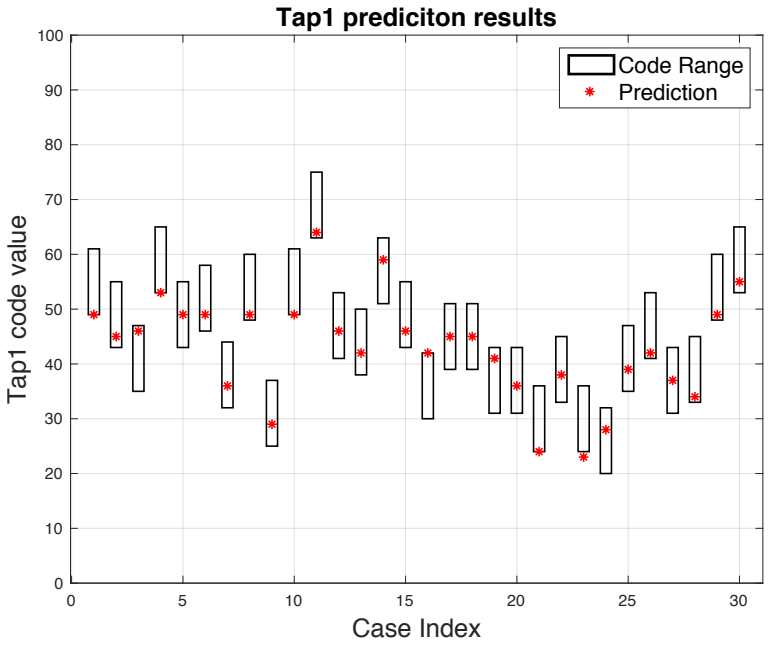
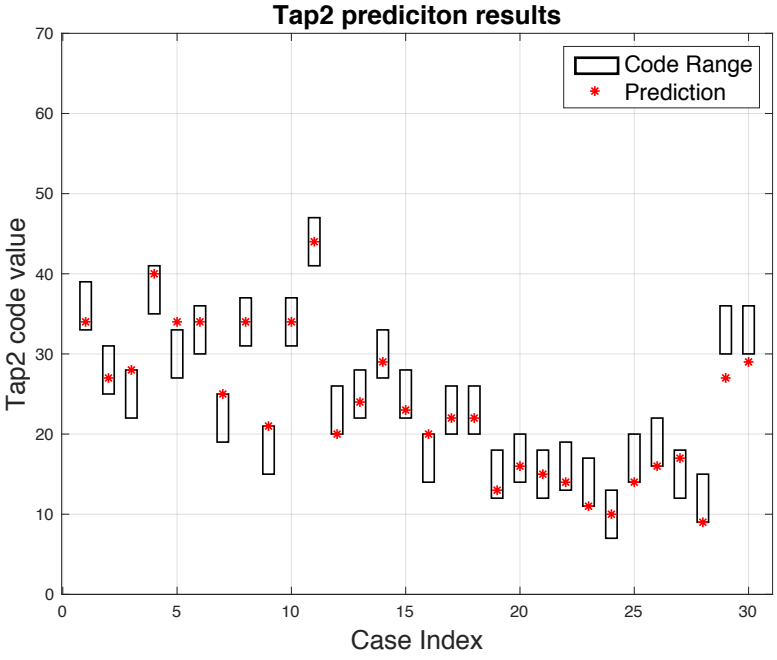
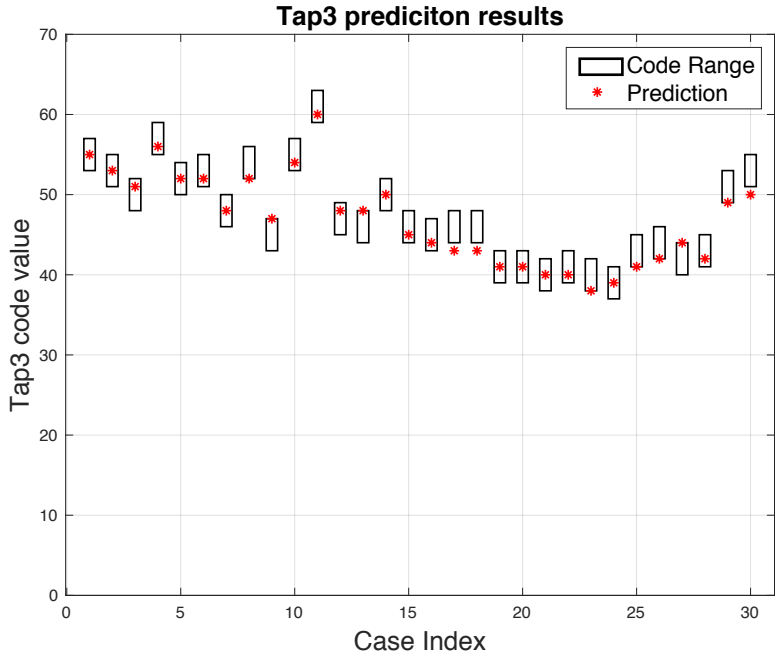

KH prediciton results
Code Range
Prediction
KH code value
Case Index
0
5
10
15
20
25
30

Figure 5.14. DFE tap1- 4 predictions in UltraScale+ GTH receiver


Tap1 prediciton results
Code Range
Prediction
Tap1 code value
Case Index
0
10
20
30
40
50
60
70
80
90
100
0
5
10
15
20
25
30


Tap2 prediciton results
Code Range
Prediction
Tap2 code value
Case Index
0
10
20
30
40
50
60
70
0
5
10
15
20
25
30


Tap3 prediciton results
Code Range
Prediction
Tap3 code value
Case Index
0
10
20
30
40
50
60
70
0
5
10
15
20
25
30


Tap4 prediciton results
Code Range
Prediction
Tap4 code value
Case Index
0
5
10
15
20
25
30
0
5
10
15
20
25
30

The prediction accuracy and worst deviation for each adaptation codes are shown in Table 5.6. The proposed model can provide high-precision adaptation prediction.

Table 5.6. Receiver adaptation code prediction accuracy in UltraScale+ GTH Transceiver

Targets	Accuracy	Worst deviation	Targets	Accuracy	Worst deviation
CTLE AGC	96.7%	1	DFE Tap5	100.0%	0
CTLE KL	90%	1	DFE Tap6	96.7%	1
CTLE KH	100%	0	DFE Tap7	93.3%	1
DFE Tap1	96.7%	1	DFE Tap8	100%	0
DFE Tap2	90%	3	DFE Tap9	96.7%	1
DFE Tap3	90%	1	DFE Tap10	96.7%	1
DFE Tap4	93.3%	1	DFE Tap11	100%	0

In this robustness test, the proposed modeling mechanism can learn different SerDes designs while still has a high correlation with the receiver adaptation behavior.

5.6. Use case of the CDL model

Once the CDL model is well trained, it can predict the receiver adaptation codes quickly instead of the IBIS-AMI simulation. Figure 5.15 shows one example of an adaptation process using the proposed CDL model vs the IBIS-AMI model simulation. The x axis is the simulation bit count. The y axis is the code value. For the IBIS-AMI model, based on the bit-by bit transient simulation, each adaptation code gradually becomes stable. However, the IBIS-AMI model would generally take 30 minutes for each adaptation simulation. With the proposed CDL model, the speed of the adaptation process significantly improves from 30 minutes to 2 seconds.

Engineers would also need to analyze the equalization status (under or over equalization) of the SerDes link according to the receiver adaptation codes and adjust TX settings accordingly.

Previously, engineers have to wait for 30 minutes to know the adaptation results. With the proposed CDL model, engineers can analyze the equalization status of the SerDes link in a short time.

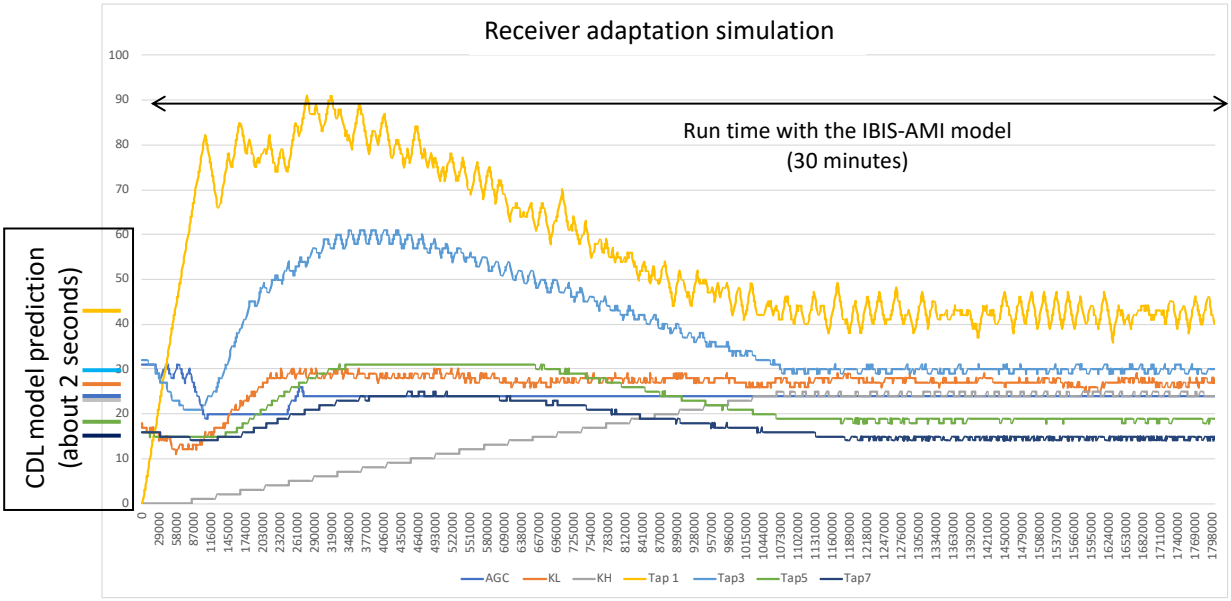


Figure 5.15. Receiver adaptation simulation using the CDL model vs. the IBIS-AMI model

5.7. Conclusion

High-speed SerDes equalization parameter auto-tuning, a.k.a. adaptation is a complex process. A new modeling mechanism, called Cascaded Deep Learning (CDL), is proposed and used in the prediction of high-speed SerDes receiver equalization adaptations. The proposed modeling method has the following capabilities:

- Cascaded prediction flow.
- Provide fast and accurate receiver adaptation simulation results.
- Learn the behaviors of different designs.

The CDL model would use a DNN model to predict CTLE adaptation codes and use separate LSTM models to predict the first three DFE taps. All the previously predicted targets are used to predict the rest DFE tap values.

To test the proposed model robustness, two different designs, namely UltraScale+ GTY and UltraScale+ GTH transceiver, are used. The CDL model can provide high-precision predictions under different channels with various TX settings for both designs. For the simulation speed, the CDL model can provide 900 times faster simulation than the state-of-the-art IBIS-AMI modeling approach.

Chapter 6. Conclusion and future work

6.1. Conclusion

With the increasing data rate and distance of the wireline communication system, signal integrity analysis becomes challenging in high-speed SerDes links. To simulate the performance of the entire SerDes link, behaviors of the transceiver need to be accurately modeled. In this work, we focus on the receiver behavioral modeling using machine learning approach.

Nowadays, the IBIS-AMI model becomes the most efficient SerDes simulator in the industry. The IBIS-AMI model can do signal integrity analysis in about 30 minutes for each channel. After each simulation, users can get the equalized transient simulation results, namely transient waveform, eye diagram and bathtub curve, and adaptation simulation results, such as adaptation codes at the receiver. However, the IBIS-AMI model can only do adaptation and transient simulation at the same time, which would be inefficient. In that case, we need to find a way to build models for these two behaviors separately, while the simulation speed of the behavioral models should be faster than the IBIS-AMI model. In this work, we use machine learning techniques to learn the receiver transient and adaptation behavior.

As for the transient behavior, a machine learning approach, named adaptive-ordered system identification model, is proposed to build high-speed receiver transient behavioral models. The proposed model can provide high-correlation simulations. The model can do transient waveform, eye diagram, and bathtub curve simulation at the same time. The training process of the model is fast and automatic. Moreover, the simulation speed of the proposed model is 10 seconds for a half-million-bit simulation, which is 180 times faster than the IBIS-AMI model.

In the adaptation modeling work, a new modeling mechanism, called Cascaded Deep Learning (CDL), is proposed and used in the prediction of high-speed SerDes receiver equalization

adaptations. The proposed modeling method has the capability to provide fast and accurate receiver adaptation simulation results. It has good scalability, which can learn the behaviors from different designs without modifying its structure. The simulation speed of the proposed model is 2 seconds for each run, which is 900 times faster than the state-of-the-art IBIS-AMI modeling approach.

6.2. Future work

The reason why we want to improve the simulation speed of the receiver adaptation process is that the receiver adaptation codes can provide useful information for the TX tuning. In the SerDes link, only the receiver has the capability to do the adaptation. The receiver can do the adaptation because it can get TX and channel information from the channel output waveform. However, the TX cannot know the channel distortion information. Currently, most of the TX settings are set manually, which is not efficient.

Recently, the backchannel method [47] could solve the TX tuning problem. Backchannel capability is the ability of the serial link RX to automatically tune the equalization settings of the serial link TX to optimize signal integrity and BER. The purpose of the backchannel method is to let TX aware of how much channel loss compensation should be according to the RX adaptation information. According to the RX adaptation codes such as the KH, KL, AGC and the DFE taps, the equalization status at RX can be decided, i.e. equalized, under equalized or, over equalized. If the SerDes link is over equalized, no TX pre-emphasis is needed. When the link is under equalized, there would be some improvement from applying TX pre-emphasis.

However, this capability only exists in one serial link standard, named PCI Express 4.0 [48]. In the PCI Express application, the RX provides suggestions according to a lookup table,

which is not efficient and cannot be used in other serial link standards. Hence, a general backchannel modeling method is needed to satisfy all the SerDes link standards.

Because of the proposed CDL model, we can get receiver adaptation codes in seconds instead of a half-hour simulation (using the IBIS-AMI model). In that case, machine learning models could be built to provide TX setting suggestions based on the simulated adaptation codes. The backchannel modeling method is shown in Figure 6.1.

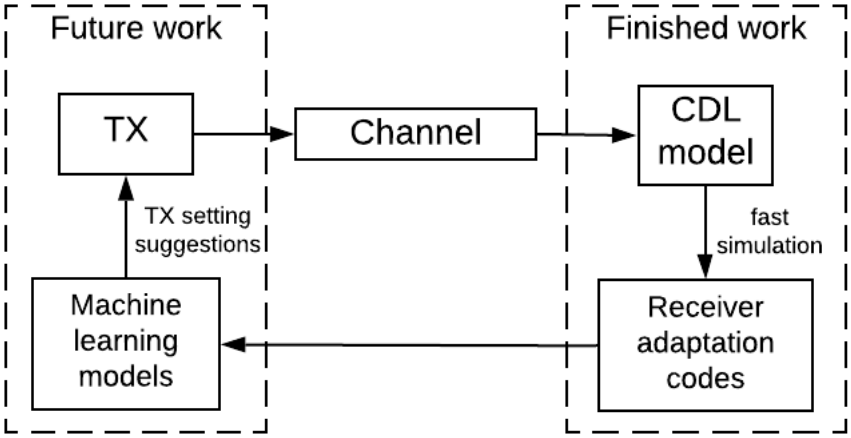


Figure 6.1. Backchannel modeling using machine learning approach

This modeling method could be applied in different SerDes standards. With this approach, the TX is also adaptive, which would improve the overall SerDes link performance.

REFERENCES

[1] Tangirala, Arun K. Principles of System Identification: Theory and Practice. CRC Press, 2014

[2] M. Norgaard, O. Ravn, N. K. Poulsen and L.K. Hansen, 'Neural Networds for Modelling and control of Dynamic Systems', 2004

[3] Hertz, J., Krogh, A., and Palmer, R. G., 'An Introduction to the Theory of Neural Computation', Lecture Notes, Volume I. Addison-Wesley, Redwood City, CA. 1991

[4] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, (USA), pp. 807–814, Omnipress, 2010

[5] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747, 2016

[6] Heng Shi, Minghao Xu, and Ran Li, 'Deep Learning for Household Load Forecasting – A Novel Pooling Deep RNN', IEEE Transactions on Smart Grid, Volume: PP, Issue: 99, 2017

[7] THESE ` N, 'Long Short-Term Memory in Recurrent Neural Networks', PhD Thesis, 2001

[8] Manish Pandey, "Machine Learning and Systems for Building the Next Generation of EDA tools", 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), 22-25 Jan. 2018

[9] Alex Manukovsky, Zurab Khasidashvili, Adam J Norman, Yaron Juniman, Roee Bloch, "Machine Learning Applications for Simulation and Modeling of 56 and 112 Gb SerDes Systems", DesignCon 2019

[10] Francisco Elias Rangel-Patiño , José Ernesto Rayas-Sánchez , Andres Viveros-Wacher , José Luis Chávez-Hurtado , Edgar Andrei Vega-Ochoa, and Nagib Hakim, "Post-Silicon Receiver Equalization Metamodeling by Artificial Neural Networks", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 38, NO. 4, APRIL 2019

[11] Tianjian Lu, Ken Wu, "Machine Learning Methods in High-Speed Channel Modeling", DesignCon 2019

[12] Thong Nguyen ; Tianjian Lu ; Ju Sun ; Quang Le ; Ken We ; Jose Schut-Aine, "Transient Simulation for High-Speed Channels with Recurrent Neural Network", 2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), 14-17 Oct. 2018

[13] B. Li, P. Franzon, Y. Choi, C. Cheng, 'Receiver Behavior Modeling based on System Identification', 2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), 14-17 Oct. 2018

[14] Y. Choi and C. Cheng, 'High-Speed Link Analysis with System Identification Approach', 2017 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI)

[15] "I/O Buffer Information Specification", Version 5.0, IBIS Open Forum

[16] Bob Sullivan, Michael Rose, Jason Boh, “Simulating High-Speed Serial Channels with IBIS-AMI Models”, Keysight Technologies Application Note

[17] Ilya Sutskever, ‘Training recurrent neural networks’, PhD Thesis, 2013.

[18] “The fall of RNN / LSTM”. https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0

[19] H. Zhang, F. Rao, T. Keulenaer, K. Ly, R. Pierco, G. Zhang, ‘IBIS-AMI Modeling and Simulation of Link Systems using Duobinary Signaling’, DesignCon, 2017

[20] IEEE Std 802.3bj-2014, AMENDMENT TO IEEE Std 802.3-2012: Ethernet.

[21] S. Grossberg. Recurrent neural networks. Scholarpedia, 8(2):1888, 2013. revision #138057.

[22] M. Chen, M. Chang ; C. Ken Yang, "A low-PDP and low-area repeater using passive CTLE for on-chip interconnects", 2015 Symposium on VLSI Circuits (VLSI Circuits), 17-19 June 2015.

[23] Sh. Melikyan, S. Sahakyan, H. Safaryan, H. Dingchyan, ”High Accuracy Equalization Method for Receiver Active Equalizer“, East-West Design & Test Symposium (EWDTS 2013), 27-30 Sept. 2013

[24] Application Report, SLLA338–June 2013, “The Benefits of Using Linear Equalization in Backplane and Cable Applications”, Texas Instruments.

[25] Yu Liao, Echo Ma, Jinhua Chen, Geoff Zhang, "25G Long Reach Cable Link System Equalization Optimization", DesignCon 2016

[26] Siegelmann, H.T., Sontag, E.D. "Turing computability with neural nets" (PDF). Appl. Math. Lett. 4 (6): 77–80, 1991

[27] D. Yu, L. Deng, G. Li, and F. Seide (2011). "Discriminative pretraining of deep neural networks," U.S. Patent Filing

[28] R. Trinchero, F. G. Canavero, “Modeling of Eye Diagram Height in High-Speed Links via Support Vector Machine”, 2018 IEEE 22nd Workshop on Signal and Power Integrity (SPI), 22-25 May 2018

[29] R. Trinchero, P. Manfredi, S. Stievano, F. G. Canavero, "Machine Learning for the Performance Assessment of High-Speed Links", IEEE Transactions on Electromagnetic Compatibility, Page(s): 1627 - 1634, 2018

[30] Anthony Sanders, Mike Resso and John D. Ambrosia. Channel Compliance Testing Utilizing Novel Statistical Eye Methodology, DesignCon 2004

[31] T. Westerhoff, M. Steinberger, W. Katz, B. Katz, A. Hawes, K. Dramstad, “Predicting BER with IBIS-AMI”, DesignCon, Feb. 4, 2010

[32] W. Katz, M. Steinberger, T. Westerhoff, “IBIS-AMI Terminology Overview”, 2009 IBIS Summit

[33] IBIS Modeling Cookbook for IBIS Version 4.0, The IBIS Open Forum, Sept. 15, 2005

[34] H. Zhang, J. Baprawski, P. Alavi, G. Zhang, "A New Methodology for Developing IBIS-AMI Models", DesignCon 2015

[35] Marcus Duelk ; Martin Zirngibl, "100 Gigabit Ethernet - Applications, Features, Challenges", Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, 23-29 April 2006

[36] J. Lee, 'A 20-Gb/s Adaptive Equalizer in 0.13-muhboxmCMOS Technology', IEEE Journal of Solid-State Circuits, 2006, Pages 2058 - 2066

[37] H. Uchiki, Y. Ota, M. Tani, Y. Hayakawa, K. Asahina, 'A 6Gb/s RX Equalizer Adapted Using Direct Measurement of the Equalizer Output Amplitude', 2008 IEEE International Solid-State Circuits Conference

[38] F. Gerfers, G. Besten, P. Petkov, J. Conder, A. Koellm, 'A 0.2–2 Gb/s 6x OSR Receiver Using a Digitally Self-Adaptive Equalizer', IEEE Journal of Solid-State Circuits, 2008, Pages 1436 – 1448

[39] R. Payne, P. Landman, B. Bhakta ; S. Ramaswamy ; S. Wu ; J.D. Powers, M.U. ErRobert Payne, P. Landman, B. Bhakta,S. Ramaswamy, Ah-Lyan Yee, R. Gu, L. Wu, Y. Xie, B. Parthasarathy, K. Brouse, W. Mohammed, K. Heragu, V. Gupta, L. Dyson, W. Lee, 'A 6.25-Gb/s binary transceiver in 0.13-/spl mu/m CMOS for serial data transmission across high loss legacy backplane channels', IEEE Journal of Solid-State Circuits, 2005, Pages 2646 - 2657

[40] K. Huang, D. Wentzloff, 'A 60GHz antenna-referenced frequency-locked loop in 0.13µm CMOS for wireless sensor networks', 2011 IEEE International Solid-State Circuits Conference

[41] B. Jiao, 'Leveraging UltraScale Architecture Transceivers for High-Speed Serial I/O Connectivity', White Paper: UltraScale GTH/GTY Transceivers, 2015

[42] 'UltraScale Architecture GTY Transceivers', User Guide, 2017

[43] Chen, Z., Raginsky, M., & Rosenbaum, E. (2017, October). Verilog-a compatible recurrent neural network model for transient circuit simulation. In 2017 IEEE 26th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS) (pp. 1-3). IEEE.

[44] Jasper Snoek, Hugo Larochelle, Ryan P. Adams, 'Practical Bayesian Optimization of Machine Learning Algorithms', Neural Information Processing Systems Conference(NIPS), 2012

[45] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," Proceedings of the IEEE, vol. 104, no. 1, pp. 148–175, 2016.

[46] J. Mocˇkus, "On Bayesian methods for seeking the extremum," in Optimization Techniques IFIP Technical Conference, 1975, pp. 400– 404.

[47] Mohammad S. Mobin et al., TX back channel adaptation algorithm and protocol emulation with application to PCIe, SAS, FC, and 10GBASE-KR, DesignCon 2012.å

[48] PCI Express Base Specification Revision 4.0 Version 1.0 September 27, 2017.

[49] Schlemper, J., Caballero, J., Hajnal, J. V., Price, A. N., & Rueckert, D. A deep cascade of convolutional neural networks for dynamic MR image reconstruction. IEEE transactions on Medical Imaging, 37(2), 491-503, 2017

[50] Marquez, E. S., Hare, J. S., & Niranjan, M. Deep cascade learning. IEEE transactions on neural networks and learning systems, 29(11), 5475-5485, 2018

[51] Levenberg, Kenneth (1944). "A Method for the Solution of Certain Non-Linear Problems in Least Squares". Quarterly of Applied Mathematics. 2 (2): 164–168.

[52] Jim Frost, “Interpreting Correlation Coefficients”, URL: https://statisticsbyjim.com/basics/correlations/, 2019

[53] Anthony Chan Carusone, "An Equalizer Adaptation Algorithm to Reduce Jitter in Binary Receivers", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, VOL. 53, NO. 9, SEPTEMBER 2006

APPENDICES

Appendix A

In this appendix, modeling scripts for the ANNARX model would be presented. The model would first do the order selection and then, do the training and testing for the given input and output data pair.

A.1 Order selection

The script listed below is for the order selection for both CTLE and DFE mode. It's implemented in Python. After analyzing the PCC score, it can tell user the best orders for the input and outputs.

```
from numpy import mean
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
import numpy as np
from pandas import DataFrame
from pandas import concat
from keras import optimizers
from sklearn.preprocessing import MinMaxScaler
import pandas
from pandas import concat
from pandas import read_csv
import numpy
from treeinterpreter import treeinterpreter as ti
from sklearn.model_selection import cross_val_score, ShuffleSplit, train_test_split
import math
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, export_graphviz
import pickle
import pydot
rx_ctle_feature_file = 'data/rx_ctle_data.csv'
rx_ctle_target_file = 'data/rx_ctle_data.csv'
rx_dfe_feature_file = 'data/rx_dfe_data.csv'
rx_dfe_target_file = 'data/rx_dfe_data.csv'
pcc_threshold = 0.6 % set PCC threshold
# ================================================================================
# load data
# input_data = read_csv('data/input/ctle_train_in.csv', header=None, names=['input_waveform'])
# output_data = read_csv('data/output/ctle_out.csv', header=None, names=['output_waveform'])
def mul(x, y):
```

```
    try:
        return pandas.to_numeric(x) * y
    except:
        return x

# ================================================================================
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, j=0, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j, i))]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % j)]
        else:
            names += [('var%d(t+%d)' % (j, i))]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
# ================================================================================
# integer encode direction
if generate_data:
    dataset = concat([input_data, output_data], axis=1)
    NUM_INPUTS = dataset.shape[1]
    NUM_OUTPUTS = 1
    # ignore headers
    input_data = input_data.values[1:, :]
    output_data = output_data[1:]
    values = dataset.values[1:]
    values = values.astype('float32')
    # normalize features
    scaler = MinMaxScaler(feature_range=(-1, 1)).fit(values)
    scaled = scaler.transform(values)

    input_reframed = series_to_supervised(input_data, TRACKING_ORDER, NUM_OUTPUTS, j=1)
    output_reframed = series_to_supervised(output_data, TRACKING_ORDER, NUM_OUTPUTS, j=2)

    # ================================================================================
    # split into train and test sets
    features = concat([input_reframed, output_reframed.iloc[:, :-1]], axis=1)
    features.dropna(inplace=True)
    target = output_reframed.iloc[:, -1]
    target.dropna(inplace=True)
```

```
    target = pandas.DataFrame(target.values, columns = ['target'])
    f_rows, f_cols = features.shape
    t_rows = target.shape
    if t_rows[0] > f_rows:
        target = target.iloc[t_rows[0] - f_rows:]

    print(features.head())
    print(target.head())

    features.to_csv(rx_ctle_feature_file, index=False)
    target.to_csv(rx_ctle_target_file, index=False)

MODEL_PATH = 'models/'
TREE_PATH = 'trees/'
NUM_INPUTS = (TRACKING_ORDER + 1) * 2 - 1
NUM_OUTPUTS = 1
TEST_PROPORTION = 0.3

def read_data(data_file):
    input_data = read_csv(data_file, header=0)
    numpy_dataset = numpy.array(input_data)
    numpy.set_printoptions(threshold=numpy.nan)
    return input_data, numpy_dataset
import math

def average(x):
    assert len(x) > 0
    return float(sum(x)) / len(x)

def pearson_def(x, y):
    assert len(x) == len(y)
    n = len(x)
    assert n > 0
    avg_x = average(x)
    avg_y = average(y)
    diffprod = 0
    xdiff2 = 0
    ydiff2 = 0
    for idx in range(n):
        xdiff = x[idx] - avg_x
        ydiff = y[idx] - avg_y
        diffprod += xdiff * ydiff
        xdiff2 += xdiff * xdiff
        ydiff2 += ydiff * ydiff

    return diffprod / math.sqrt(xdiff2 * ydiff2)

% analyze CTLE mode data
features = read_csv(rx_ctle_feature_file, header=0)
target = read_csv(rx_ctle_target_file, header=0)
f_rows, f_cols = features.shape
t_rows = target.shape[0]
pearson_coeff = []
for i in range(f_cols):
    pearson_coeff.append(abs(pearson_def(features.iloc[:, i].values.tolist(), target.iloc[:, 0].values.tolist())))
%matplotlib notebook
```

```
pyplot.figure()
pyplot.plot(features.iloc[:, 1023].values.tolist())
pyplot.plot(target.iloc[:, 0].values.tolist())

pyplot.figure()
input_coeff = pearson_coeff[:TRACKING_ORDER + 1]
output_coeff = pearson_coeff[TRACKING_ORDER + 1:]

# plot CTLE input influence
pyplot.plot(np.arange(-len(input_coeff) + 1, 1), input_coeff, '.-')
pyplot.title('Previous input importance to the current output in LP mode', loc='center', fontsize=15)
pyplot.xlabel('Previous Inputs', fontsize=15)
pyplot.ylabel('ABS(PCC)', fontsize=15)
pyplot.grid(True)
# pyplot.bar(np.arange(len(input_coeff)),input_coeff)

# plot CTLE output influence
pyplot.show()
pyplot.figure()
print(len(output_coeff))
pyplot.plot(np.arange(-len(output_coeff), 0), output_coeff, '.-')
pyplot.title('Previous output importance to the current output in LP mode', loc='center', fontsize=15)
pyplot.xlabel('Previous Outputs', fontsize=15)
pyplot.ylabel('ABS(PCC)', fontsize=15)
pyplot.grid(True)
# pyplot.bar(np.arange(len(output_coeff)),output_coeff)
pyplot.show()

% analyze DFE mode data
dfe_features = read_csv(rx_dfe_feature_file, header=0)
dfe_target = read_csv(rx_dfe_target_file, header=0)
f_rows, f_cols = dfe_features.shape
t_rows = dfe_target.shape[0]
dfe_pearson_coeff = []
for i in range(f_cols):
    dfe_pearson_coeff.append(abs(pearson_def(dfe_features.iloc[:, i].values.tolist(), dfe_target.iloc[:, 0].valu
es.tolist())))
%matplotlib notebook
pyplot.figure()
pyplot.plot(dfe_features.iloc[:, 1023].values.tolist())
pyplot.plot(dfe_target.iloc[:, 0].values.tolist())

dfe_input_coeff = dfe_pearson_coeff[:TRACKING_ORDER + 1]
dfe_output_coeff = dfe_pearson_coeff[TRACKING_ORDER + 1:]

# plot DFE input influence
pyplot.figure()
pyplot.plot(np.arange(-len(dfe_input_coeff) + 1, 1), dfe_input_coeff, '.-')
pyplot.title('Previous input importance to the current output in DFE mode', loc='center', fontsize=15)
pyplot.xlabel('Previous Inputs', fontsize=15)
pyplot.ylabel('ABS(PCC)', fontsize=15)
pyplot.grid(True)
pyplot.show()
# pyplot.bar(np.arange(len(input_coeff)),input_coeff)

# plot DFE output influence
```

```
198.
199.pyplot.figure()
200.pyplot.plot(np.arange(-len(dfe_output_coeff), 0), dfe_output_coeff, '.-')
201.pyplot.title('Previous output importance to the current output in DFE mode', loc='center', fontsize=15)
202.pyplot.xlabel('Previous Outputs', fontsize=15)
203.pyplot.ylabel('ABS(PCC)', fontsize=15)
204.pyplot.grid(True)
205.# pyplot.bar(np.arange(len(output_coeff)),output_coeff)
206.pyplot.show()
```

A.2 Model training script

The ANNARX model training process is implemented in Matlab. User can call this function to train the model. Details about how to run the function are shown in the comments.

```
1.  function [accuracy, pred_out, NetDef,NN,W1,W2] = nnarx_modeling(A, B, num_neurons, in_data, out_dat
    a, draw_prediction)
2.  %  nnarx_modeling
3.      %  -----
4.      %          use input and output data to build NNARX model
5.      %
6.      %  CALL:
7.      %      [accuracy, pred_out, NetDef,NN,W1,W2] = nnarx_modeling(A, B, num_neurons, in_data, out_dat
    a, draw_prediction)
8.      %
9.      %  INPUTS:
10.     %      A: the order of output selected using Python script
11.     %      B: the order of input selected using Python script
12.     %      num_neurons: how many neurons in each hidden layer
13.     %      in_data: input data
14.     %      out_data: output data
15.     %      draw_prediction: whether draw predicted output
16.
17.     %  OUTPUTS:
18.     %      accuracy: prediction accuracy
19.     %      pred_out: predicted output
20.     %      NetDef: network layer structure
21.     %      NN: nerons in each layer
22.     %      W1: weights in the first hidden layer
23.     %      W2: weights in the second hidden layer
24.
25. %  modeling data preperation
26. delay = finddelay(in_data, out_data);
27. if delay < 0
28.     delay = 0;
29. end
30.
31. trparms = settrain;
32. trparms = settrain(trparms,'maxiter', 100, 'D', 1e-4,'skip',100);
33. NetDef = [strcat('L', repmat('H', 1, num_neurons)); strcat('L', repmat('-', 1, num_neurons))];
34. u1 = in_data';
35. original = out_data';
```

```
NN = [A B delay];
[W1,W2, ~, ~,~]=nnarx(NetDef,NN,[],[], trparms, original,u1);
c=fix(clock);
fprintf('\nPrediction starts at %2i.%2i.%2i\n\n',c(4),c(5),c(6));
[pred_out, ~] = nnvalid('nnarmax2',NetDef,NN, W1,W2, original, u1);
c=fix(clock);
fprintf('\nPrediction starts at %2i.%2i.%2i\n\n',c(4),c(5),c(6));
[original, pred_out] = align_data(original, pred_out);

% save model into rx_model folder
if ~exist('rx_model', 'dir')
    mkdir rx_model;
end
if ~exist('rx_model/dfe_model', 'dir')
    mkdir rx_model/dfe_model;
end
save(sprintf('rx_model/dfe_model/%s_dfe.mat', date), 'NetDef', 'NN', 'W1' ,'W2');
% find delay again

accuracy = accuracyRate('DFE SBR', original, pred_out);

% plot transient waveform prediction result
if draw_prediction
    figure;
    plot(original, 'LineWidth', 1.5); hold on; plot(pred_out, '.', 'LineWidth', 1.5)
    title(sprintf('Accuracy = %.2f%%', accuracy), 'FontSize', 20);
    xlabel('Samples', 'FontSize', 20);
    ylabel('Amptitude (V)', 'FontSize', 20);
    ylim([1.2*min(original), 1.2*max(original)]);
    grid;
    lgnd = legend('Real Waveform', 'Predicted Waveform');
    set(lgnd,'FontSize',15);
    drawnow;
end
```

In this training script, it calls two functions, nnarx.m and marq.m, which are the main part for the model training process. Details about how to run the function are shown in the comments.

nnarx.m script:

```
function [W1,W2,PI_vector,iteration,lambda]=nnarx(NetDef,NN,W1,W2,trparms,Y,U)
%  NNARX
%  -----
%        Determine a nonlinear ARX model of a dynamic system by training a
%        two-layer neural network with the Marquardt method. The function
%        can handle multi-input systems (MISO).
%
%  CALL:
%       [W1,W2,critvec,iteration,lambda]=nnarx(NetDef,NN,W1,W2,trparms,Y,U)
%
%  INPUTS:
```

```
% U      : Input signal (= control signal) (left out in the nnar case).
%          dim(U) = [(inputs) * (# of data)]
% Y      : Output signal. dim(Y) = [1 * # of data]
% NN     : NN=[na nb nk].
%          na = # of past outputs used for determining the prediction
%          nb = # of past inputs used for determining prediction
%          nk = time delay (usually 1)
%          For multi-input systems nb and nk contain as many columns as
%          there are inputs.
% W1,W2  : Input-to-hidden layer and hidden-to-output layer weights.
%          If they are passed as [] they are initialized automatically
% trparms:  Data structure with parameters associated with the
%          training algorithm (optional). Use the function SETTRAIN if
%          you do not want to use the default values.

% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  INITIALIZATIONS  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
N  = length(Y);
na = NN(1);
if length(NN)==1
  nb = 0;                       % nnar model
  nk = 0;
  nu = 0;
else
  [nu,N]     = size(U);
  nb = NN(2:1+nu);              % nnarx model
  nk = NN(2+nu:1+2*nu);
end

nmax       = max([na,nb+nk-1]);
nab        = na+sum(nb);

% -- Initialize weights if nescessary --
if isempty(W1)| isempty(W2),
  hidden = length(NetDef(1,:));    % Number of hidden neurons
  W1 = rand(hidden,nab+1)-0.5;
  W2 = rand(1,hidden+1)-0.5;
end

% -- Initialize 'trparms' if nescessary --
if isempty(trparms), trparms=[]; end

% >>>>>>>>>>>>>>>>>>>>>>>  CONSTRUCT THE REGRESSION MATRIX PHI  <<<<<<<<<<<<<<<<<<<<<<<
PHI = zeros(nab,N-nmax);
jj  = nmax+1:N;
for k = 1:na, PHI(k,:)    = Y(jj-k); end
index = na;
for kk = 1:nu,
  for k = 1:nb(kk), PHI(k+index,:) = U(kk,jj-k-nk(kk)+1); end
  index = index + nb(kk);
end

% >>>>>>>>>>>>>>>>>>>>>>>         CALL TRAINING FUNCTION        <<<<<<<<<<<<<<<<<<<<<<<
[W1,W2,PI_vector,iteration,lambda]=marq(NetDef,W1,W2,PHI,Y(nmax+1:N),trparms);
```

marq.m script:

```
function [W1,W2,PI_vector,iteration,lambda]=marq(NetDef,W1,W2,PHI,Y,trparms)
% MARQ
% ----
%           Train a two layer neural network with the Levenberg-Marquardt
%           method.
%
%       If desired, it is possible to use regularization by
%       weight decay. Also pruned (ie. not fully connected) networks can
%       be trained.
%
%       Given a set of corresponding input-output pairs and an initial
%       network,
%       [W1,W2,critvec,iteration,lambda]=marq(NetDef,W1,W2,PHI,Y,trparms)
%       trains the network with the Levenberg-Marquardt method.
%
%       The activation functions can be either linear or tanh. The
%       network architecture is defined by the matrix 'NetDef' which
%       has two rows. The first row specifies the hidden layer and the
%       second row specifies the output layer.
%
%       E.g.:   NetDef = ['LHHHH'
%                         'LL---']
%       (L = Linear, H = tanh)
%
%       A weight is pruned by setting it to zero.
%
%       The Marquardt method is described in:
%       K. Madsen: 'Optimering' (Haefte 38), IMM, DTU, 1991
%
%       Notice that the bias is included as the last column in the weight
%       matrices.
%
%
% INPUT:
% NetDef : Network definition .
% W1     : Input-to-hidden layer weights. The matrix dimension is
%          [(# of hidden units)-by-(inputs + 1)] (the 1 is due to the bias).
%          Use [] for a random initialization.
% W2     : hidden-to-output layer weights. Dimension is
%          [(outputs)  *  (# of hidden units + 1)].
%          Use [] for a random initialization.
% PHI    : Input vector. dim(PHI) = [(inputs)  *  (# of data)].
% Y      : Output data. dim(Y) = [(outputs)  * (# of data)].
% trparms: Data structure with parameters associated with the
%          training algorithm (optional). Use the function SETTRAIN if
%          you do not want to use the default values.
%
% OUTPUT:
% W1, W2   : Weight matrices after training.
% critvec:   Vector containing the criterion evaluated at each iteration
% iteration: # of iterations
% lambda   : The final value of lambda. Relevant only if retraining is desired
```

```
53.
54.  %-------------------------------------------------------------------------------
55.  %--------------           NETWORK INITIALIZATIONS                -------------
56.  %-------------------------------------------------------------------------------
57.  [outputs,N] = size(Y);                % # of outputs and # of data
58.  [inputs,N] = size(PHI);                % # of hidden units
59.  L_hidden = find(NetDef(1,:)=='L')';    % Location of linear hidden neurons
60.  H_hidden = find(NetDef(1,:)=='H')';    % Location of tanh hidden neurons
61.  L_output = find(NetDef(2,:)=='L')';    % Location of linear output neurons
62.  H_output = find(NetDef(2,:)=='H')';    % Location of tanh output neurons
63.  hidden = length(L_hidden)+length(H_hidden);
64.  if isempty(W1) | isempty(W2),          % Initialize weights if nescessary
65.    W1 = rand(hidden,inputs+1)-0.5;
66.    W2 = rand(outputs,hidden+1)-0.5;
67.  end
68.  if (size(W1,2)~=inputs+1 | size(W1,1)~=hidden |... % Check dimensions
69.        size(W2,2)~=hidden+1 | size(W2,1)~=outputs)
70.     error('Dimension mismatch in weights, data, or NetDef.');
71.  end
72.  y1       = [zeros(hidden,N);ones(1,N)]; % Hidden layer outputs
73.  y2       = zeros(outputs,N);           % Network output
74.  index = outputs*(hidden+1) + 1 + [0:hidden-1]*(inputs+1); % A useful vector!
75.  index2 = (0:N-1)*outputs;              % Yet another useful vector
76.  iteration = 1;                        % Counter variable
77.  dw       = 1;                         % Flag telling that the weights are new
78.  PHI      = [PHI;ones(1,N)];           % Augment PHI with a row containg ones
79.  parameters1= hidden*(inputs+1);        % # of input-to-hidden weights
80.  parameters2= outputs*(hidden+1);       % # of hidden-to-output weights
81.  parameters = parameters1 + parameters2; % Total # of weights
82.  PSI      = zeros(parameters,outputs*N); % Deriv. of each output w.r.t. each weight
83.  ones_h   = ones(hidden+1,1);           % A vector of ones
84.  ones_i   = ones(inputs+1,1);           % Another vector of ones
85.                                        % Parameter vector containing all weights
86.  theta = [reshape(W2',parameters2,1) ; reshape(W1',parameters1,1)];
87.  theta_index = find(theta);             % Index to weights<>0
88.  theta_red = theta(theta_index);        % Reduced parameter vector
89.  reduced  = length(theta_index);        % The # of parameters in theta_red
90.  index3   = 1:(reduced+1):(reduced^2);   % A third useful vector
91.  lambda_old = 0;
92.  if nargin<6 | isempty(trparms) % Default training parameters
93.    trparms = settrain;
94.    lambda  = trparms.lambda;
95.    D       = trparms.D;
96.  else                                  % User specified values
97.    if ~isstruct(trparms),
98.       error('"trparms" must be a structure variable.');
99.    end
100. if ~isfield(trparms,'infolevel')
101.    trparms = settrain(trparms,'infolevel','default');
102. end
103. if ~isfield(trparms,'maxiter')
104.    trparms = settrain(trparms,'maxiter','default');
105. end
106. if ~isfield(trparms,'critmin')
107.    trparms = settrain(trparms,'critmin','default');
108. end
```

```
109.  if ~isfield(trparms,'critterm')
110.     trparms = settrain(trparms,'critterm','default');
111.  end
112.  if ~isfield(trparms,'gradterm')
113.     trparms = settrain(trparms,'gradterm','default');
114.  end
115.  if ~isfield(trparms,'paramterm')
116.     trparms = settrain(trparms,'paramterm','default');
117.  end
118.  if ~isfield(trparms,'lambda')
119.     trparms = settrain(trparms,'lambda','default');
120.  end
121.  lambda    = trparms.lambda;
122.  if ~isfield(trparms,'D')
123.     trparms = settrain(trparms,'D','default');
124.     D = trparms.D;
125.  else
126.    if length(trparms.D)==1,              % Scalar weight decay parameter
127.       D = trparms.D(ones(1,reduced));
128.    elseif length(trparms.D)==2,          % Two weight decay parameters
129.       D = trparms.D([ones(1,parameters2) 2*ones(1,parameters1)])';
130.       D = D(theta_index);
131.    elseif length(trparms.D)>2,           % Individual weight decay
132.       D = trparms.D(:);
133.    end
134.  end
135.end
136.D = D(:);
137.critdif  = trparms.critterm+1;            % Initialize stopping variables
138.gradmax  = trparms.gradterm+1;
139.paramdif = trparms.paramterm+1;
140.PI_vector = zeros(trparms.maxiter,1);     % Vector for storing criterion values
141.
142.
143.%-------------------------------------------------------------------------------
144.%--------------                TRAIN NETWORK                  -------------
145.%-------------------------------------------------------------------------------
146.% clc;
147.% c=fix(clock);
148.% fprintf('Network training started at %2i.%2i.%2i\n\n',c(4),c(5),c(6));
149.
150.
151.% >>>>>>>>>>>>>>>>>>>>>>>  COMPUTE NETWORK OUTPUT  y2(theta)   <<<<<<<<<<<<<<<<<<<
      <<<<
152.h1 = W1*PHI;
153.y1(H_hidden,:) = pmntanh(h1(H_hidden,:));
154.y1(L_hidden,:) = h1(L_hidden,:);
155.
156.h2 = W2*y1;
157.y2(H_output,:) = pmntanh(h2(H_output,:));
158.y2(L_output,:) = h2(L_output,:);
159.
160.E        = Y - y2;                  % Training error
161.E_vector = E(:);                    % Reshape E into a long vector
162.SSE      = E_vector'*E_vector;       % Sum of squared errors (SSE)
163.PI       = (SSE+theta_red'*(D.*theta_red))/(2*N); % Performance index
```

```
164.
165.% Iterate until stopping criterion is satisfied
166.while (iteration<=trparms.maxiter & PI>trparms.critmin & lambda<1e7 & ...
167.       (critdif>trparms.critterm | gradmax>trparms.gradterm | ...
168.       paramdif>trparms.paramterm))
169.
170.    if dw==1,
171.    % >>>>>>>>>>>>>>>>>>>>>>>>>>>>   COMPUTE THE PSI MATRIX   <<<<<<<<<<<<<<<<<<<<<<<<
       <<<
172.    % (The derivative of each network output (y2) with respect to each weight)
173.
174.     % ==========   Elements corresponding to the linear output units   ============
175.     for i = L_output'
176.       index1 = (i-1) * (hidden + 1) + 1;
177.
178.       % -- The part of PSI corresponding to hidden-to-output layer weights --
179.       PSI(index1:index1+hidden,index2+i) = y1;
180.       % ------------------------------------------------------------------
181.
182.       % -- The part of PSI corresponding to input-to-hidden layer weights ---
183.       for j = L_hidden',
184.         PSI(index(j):index(j)+inputs,index2+i) = W2(i,j)*PHI;
185.       end
186.
187.       for j = H_hidden',
188.         tmp = W2(i,j)*(1-y1(j,:).*y1(j,:));
189.         PSI(index(j):index(j)+inputs,index2+i) = tmp(ones_i,:).*PHI;
190.       end
191.       % ------------------------------------------------------------------
192.     end
193.
194.     % ============  Elements corresponding to the tanh output units   ==============
195.     for i = H_output',
196.       index1 = (i-1) * (hidden + 1) + 1;
197.
198.       % -- The part of PSI corresponding to hidden-to-output layer weights --
199.       tmp = 1 - y2(i,:).*y2(i,:);
200.       PSI(index1:index1+hidden,index2+i) = y1.*tmp(ones_h,:);
201.       % ------------------------------------------------------------------
202.
203.       % -- The part of PSI corresponding to input-to-hidden layer weights ---
204.       for j = L_hidden',
205.         tmp = W2(i,j)*(1-y2(i,:).*y2(i,:));
206.         PSI(index(j):index(j)+inputs,index2+i) = tmp(ones_i,:).*PHI;
207.       end
208.
209.       for j = H_hidden',
210.         tmp  = W2(i,j)*(1-y1(j,:).*y1(j,:));
211.         tmp2 = (1-y2(i,:).*y2(i,:));
212.         PSI(index(j):index(j)+inputs,index2+i) = tmp(ones_i,:)...
213.                                                  .*tmp2(ones_i,:).*PHI;
214.       end
215.       % ------------------------------------------------------------------
216.     end
217.     PSI_red = PSI(theta_index,:);
218.
```

```
  % -- Gradient --
  G = PSI_red*E_vector-D.*theta_red;

  % -- Means square error part Hessian  --
  H = PSI_red*PSI_red';
  H(index3) = H(index3)'+D;                        % Add diagonal matrix
  dw = 0;
end


% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>        COMPUTE h_k        <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
% -- Hessian  --
H(index3) = H(index3)'+(lambda-lambda_old);       % Add diagonal matrix

% -- Search direction --
h = H\G;                                          % Solve for search direction

% -- Compute 'apriori' iterate --
theta_red_new = theta_red + h;                    % Update parameter vector
theta(theta_index) = theta_red_new;

% -- Put the parameters back into the weight matrices --
W1_new = reshape(theta(parameters2+1:parameters),inputs+1,hidden)';
W2_new = reshape(theta(1:parameters2),hidden+1,outputs)';


% >>>>>>>>>>>>>>>>>>>>>>>>   COMPUTE NETWORK OUTPUT  y2(theta+h)   <<<<<<<<<<<<<<<<<<<<<
h1 = W1_new*PHI;
y1(H_hidden,:) = pmntanh(h1(H_hidden,:));
y1(L_hidden,:) = h1(L_hidden,:);

h2 = W2_new*y1;
y2(H_output,:) = pmntanh(h2(H_output,:));
y2(L_output,:) = h2(L_output,:);

E_new        = Y - y2;                  % Training error
E_new_vector = E_new(:);                % Reshape E into a long vector
SSE_new  = E_new_vector'*E_new_vector; % Sum of squared errors (SSE)
PI_new   = (SSE_new + theta_red_new'*(D.*theta_red_new))/(2*N); % PI


% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>       UPDATE  lambda     <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
L = h'*G + h'*(h.*(D+lambda));
lambda_old = lambda;

% Decrease lambda if SSE has fallen 'sufficiently'
if 2*N*(PI - PI_new) > (0.75*L),
  lambda = lambda/2;

% Increase lambda if SSE has grown 'sufficiently'
elseif 2*N*(PI-PI_new) <= (0.25*L),
  lambda = 2*lambda;
end


```

```
% >>>>>>>>>>>>>>>>>>>>>>>>      UPDATES FOR NEXT ITERATION      <<<<<<<<<<<<<<<<<<<<<<<<
 % Update only if criterion has decreased
 if PI_new < PI,
   critdif  = PI-PI_new;                    % Criterion difference
   gradmax  = max(abs(G))/N;                % Maximum gradient
   paramdif = max(abs(theta_red_new - theta_red)); % Maximum parameter dif.
   W1 = W1_new;
   W2 = W2_new;
   theta_red = theta_red_new;
   E_vector = E_new_vector;
   PI = PI_new;
   dw = 1;
   lambda_old = 0;
   iteration = iteration + 1;
   PI_vector(iteration-1) = PI;                % Collect PI in vector
%    switch(trparms.infolevel)                  % Print on-line inform
%       case 1
%          fprintf('# %i   W=%4.3e  critdif=%3.2e  maxgrad=%3.2e  paramdif=%3.2e\n',...
%                                        iteration-1,PI,critdif,gradmax,paramdif);
%       otherwise
%          fprintf('iteration # %i   W = %4.3e\r',iteration-1,PI);
%    end
 end
end
%-------------------------------------------------------------------------------
%--------------          END OF NETWORK TRAINING             ----------
%-------------------------------------------------------------------------------
iteration = iteration-1;
PI_vector = PI_vector(1:iteration);
% c=fix(clock);
% fprintf('\n\nNetwork training ended at %2i.%2i.%2i\n',c(4),c(5),c(6));
```

A.3 Model testing script

Once the model is well-trained, users can use testing data to test the model performance in transient waveform, eye diagram and bathtub curve. Details about how to run the function are shown in the comments.

```
function [pred_out, accuracy] = nnarx_testing(NetDef,NN,W1,W2, input_data, output_data, fs, SamplesPerSymbol, draw_prediction, draw_eye)
%  nnarx_testing
    %  -----
    %         test NNARX model
    %
    %  CALL:
    %      [pred_out, accuracy] = nnarx_testing(NetDef,NN,W1,W2, input_data, output_data, fs, SamplesPerSymbol, draw_prediction, draw_eye)
    %
    %  INPUTS:
    %  NetDef: network layer structure
```

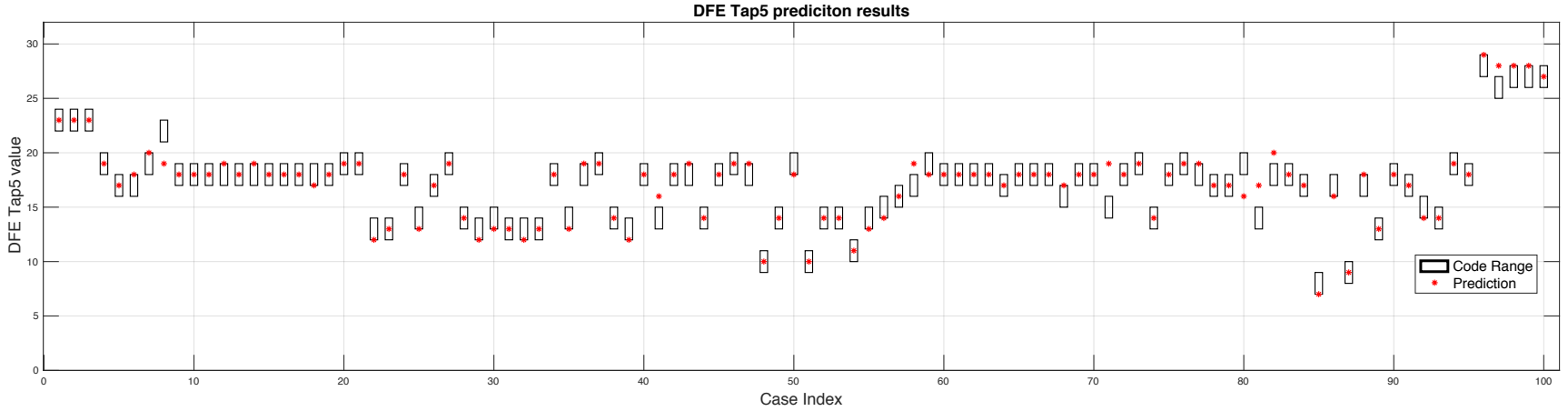
```
    %  NN: nerons in each layer
    %  W1,W2   : Input-to-hidden layer and hidden-to-output layer weights.
    %            If they are passed as [] they are initialized automatically
    %  input_data: input data
    %  output_data: ground-truth output data
    %  fs: sampling frequency
    %  SamplesPerSymbol: Samples Per Symbol
    %  draw_prediction: whether to draw prediction output
    %  draw_eye: whether to draw predicted eye diagram and bathtub curve
    %
    %  OUTPUTS:
    %       pred_out: predicted output
    %       accuracy: prediction accuracy
    %
ber_level = 15;  % BER level for the bathtub curve
u1 = input_data';
original = output_data';

c=fix(clock);
fprintf('\nPrediction starts at %2i.%2i.%2i\n\n',c(4),c(5),c(6));
[pred_out, ~] = nnvalid('nnarmax2',NetDef,NN, W1,W2, original, u1);
% pred_out=nnsimul('nnarmax2',NetDef,NN,W1,W2,original,u1);
c=fix(clock);
fprintf('\nPrediction starts at %2i.%2i.%2i\n\n',c(4),c(5),c(6));
% find delay again
out_delay = finddelay(original, pred_out);
if out_delay < 0
    original = original(1 + abs(out_delay): end);
    pred_out = pred_out(1: end + out_delay);
else
    pred_out = pred_out(1 + out_delay: end);
    original = original(1: end - out_delay);
end
[original, pred_out] = align_data(original, pred_out);


original = original - mean(original);
pred_out = pred_out - mean(pred_out);

% ignore spike
for i = 2: length(pred_out) - 1
    if abs(pred_out(i) - pred_out(i - 1)) > 0.03 &&  abs(pred_out(i) - pred_out(i + 1)) > 0.015
        pred_out(i) = pred_out(i + 1);
    end
end

disp('Tran model testing results:');
accuracy = accuracyRate('DFE', original, pred_out);

% plot transient waveform prediction result
if draw_prediction
    figure;
    plot(original, 'LineWidth', 1.5); hold on; plot(pred_out, '--', 'LineWidth', 1.5)
    title(sprintf('Prediction accuracy = %.3f%%', accuracy), 'FontSize', 20);
    xlabel('Samples', 'FontSize', 20);
    ylabel('Amptitude (V)', 'FontSize', 20);
```

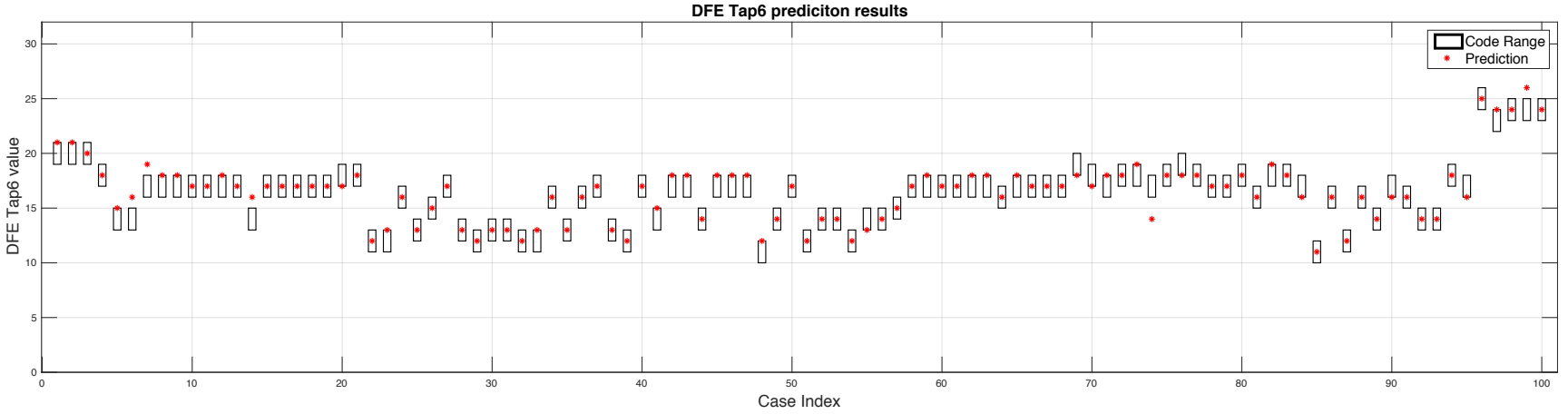
```
    ylim([1.2*min(original), 1.2*max(original)]);
    grid;
    lgnd = legend('Real Waveform', 'Predicted Waveform');
    set(lgnd,'FontSize',15);
    drawnow;
end

% plot eye prediction result
if draw_eye
    disp('Plot CTLE output real eye...');
    ed = comm.EyeDiagram('YLimits', [1.2*min(original) - 0.1, 1.2*max(original)], ...
        'SamplesPerSymbol', SamplesPerSymbol, ...
        'SampleRate', fs, ...
        'DisplayMode', '2D color histogram', ...
        'ColorScale', 'Logarithmic', ...
        'EnableMeasurements',true,...
        'BathtubBER', [0.5 10.^-(1:ber_level)], ...
        'ShowBathtub','Both');
    ed(original');

    disp('Plot CTLE output predict eye...');
    pred_ed = comm.EyeDiagram('YLimits', [1.2*min(original) - 0.1, 1.2*max(original)], ...
        'SamplesPerSymbol', SamplesPerSymbol, ...
        'SampleRate', fs, ...
        'DisplayMode', '2D color histogram', ...
        'ColorScale', 'Logarithmic', ...
        'EnableMeasurements',true,...
        'BathtubBER', [0.5 10.^-(1:ber_level)], ...
        'ShowBathtub','Both');
    pred_ed(pred_out');
    disp('Finish CTLE modeling.');

     figure();
    hb = horizontalBathtub(ed);
    semilogy([hb.LeftThreshold],[hb.BER],'b',[hb.RightThreshold],[hb.BER],'b', 'LineWidth', 2); hold on;
    hb_pred = horizontalBathtub(pred_ed);
    semilogy([hb_pred.LeftThreshold],[hb_pred.BER],'r--',[hb_pred.RightThreshold],[hb_pred.BER],'r--
', 'LineWidth', 2);
    title('Horizontal Bathtub Curve comparison', 'FontSize', 20);
    xlabel('Time', 'FontSize', 20); ylabel('BER', 'FontSize', 20);
     legend('Real Curve (left)','Real Curve (right)', 'Predicted Curve (left)','Predicted Curve (right)',  'FontSiz
e', 15);
    grid
    y = zeros(length(hb), 1);
    y_hat = zeros(length(hb), 1);

    for i = 1: length(hb)
        y(i) = abs(hb(i).RightThreshold - hb(i).LeftThreshold);
        y_hat(i) = abs(hb_pred(i).RightThreshold - hb_pred(i).LeftThreshold);
    end
    RMSE_h = sqrt(mean((y - y_hat).^2));
    worst_h = sqrt(mean((y(1) - y_hat(1)).^2));

    figure();
    vb = verticalBathtub(ed);
```

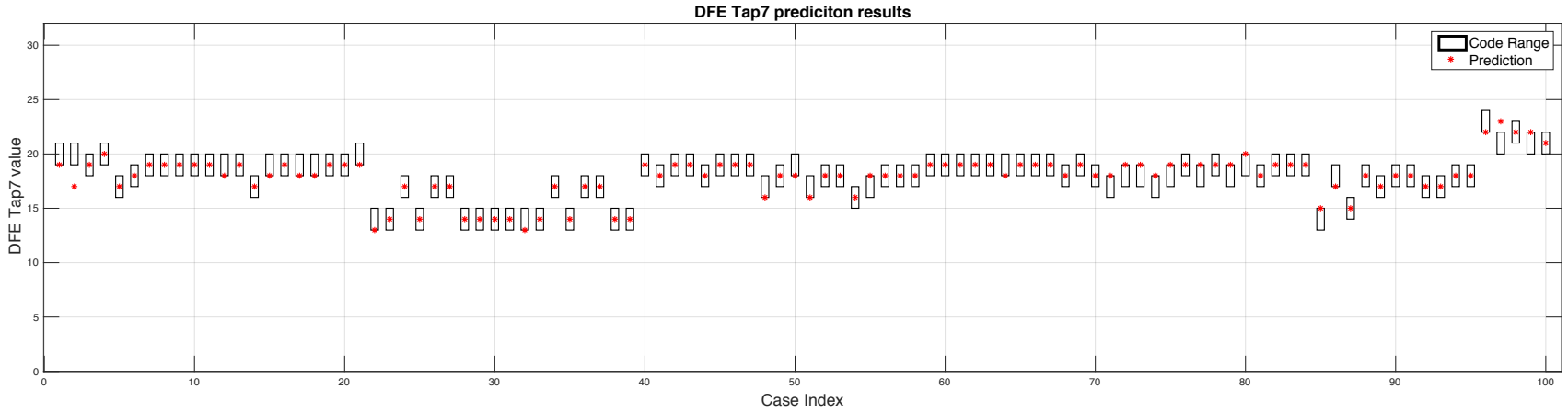
```
120.    semilogx([vb.BER],[vb.LowerThreshold],'b' ,[vb.BER],[vb.UpperThreshold],'b', 'LineWidth', 2); hold on
    ;
121.    vb_pred = verticalBathtub(pred_ed);
122.    semilogx([vb_pred.BER],[vb_pred.LowerThreshold],'r--' ,[vb_pred.BER],[vb_pred.UpperThreshold],'r--
    ', 'LineWidth', 2);
123.     title('Vertical Bathtub Curve comparison', 'FontSize', 20);
124.    xlabel('BER', 'FontSize', 20); ylabel('Amplitude', 'FontSize', 20);
125.    legend('Real Curve (lower)','Real Curve (upper)', 'Predicted Curve (lower)','Predicted Curve (upper)',  'Fo
    ntSize', 15);
126.    grid
127.
128.
129.    y = zeros(length(vb), 1);
130.    y_hat = zeros(length(vb), 1);
131.
132.    for i = 1: length(vb)
133.        y(i) = abs(vb(i).UpperThreshold - vb(i).LowerThreshold);
134.        y_hat(i) = abs(vb_pred(i).UpperThreshold - vb_pred(i).LowerThreshold);
135.    end
136.    RMSE_v = sqrt(mean((y - y_hat).^2));
137.    worst_v = sqrt(mean((y(1) - y_hat(1)).^2));
138.    fprintf('Horizontal Bathtub prediction RMSE: %e\n', RMSE_h);
139.    fprintf('Horizontal Bathtub prediction Worst Error: %e\n', worst_h);
140.    fprintf('Vertical Bathtub prediction RMSE: %e\n', RMSE_v);
141.    fprintf('Vertical Bathtub prediction Worst Error: %e\n', worst_v);
142.    disp('Finish DFE prediction.');
143.end
```

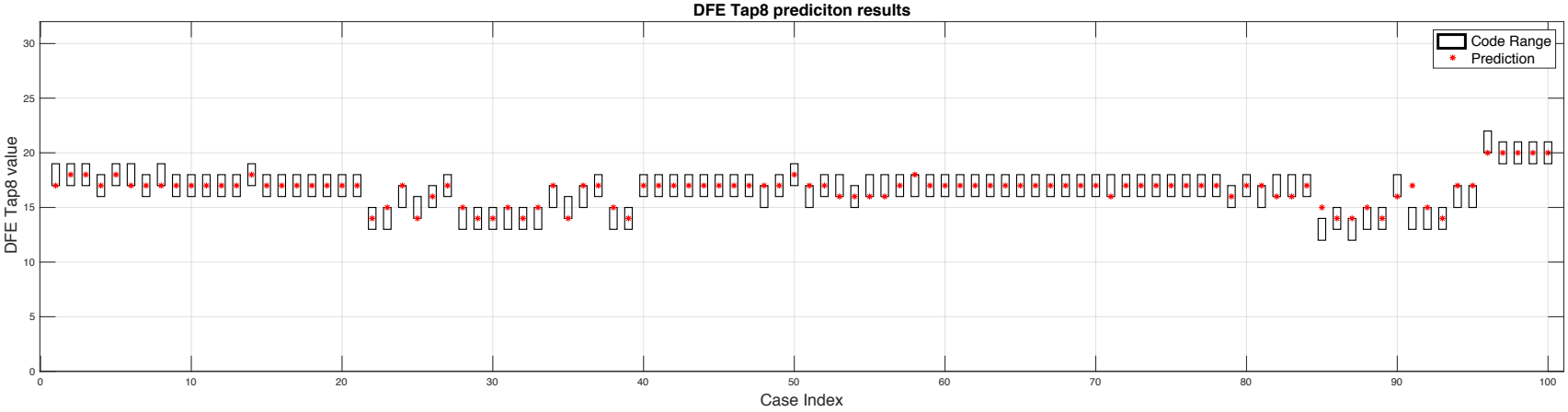
Appendix B

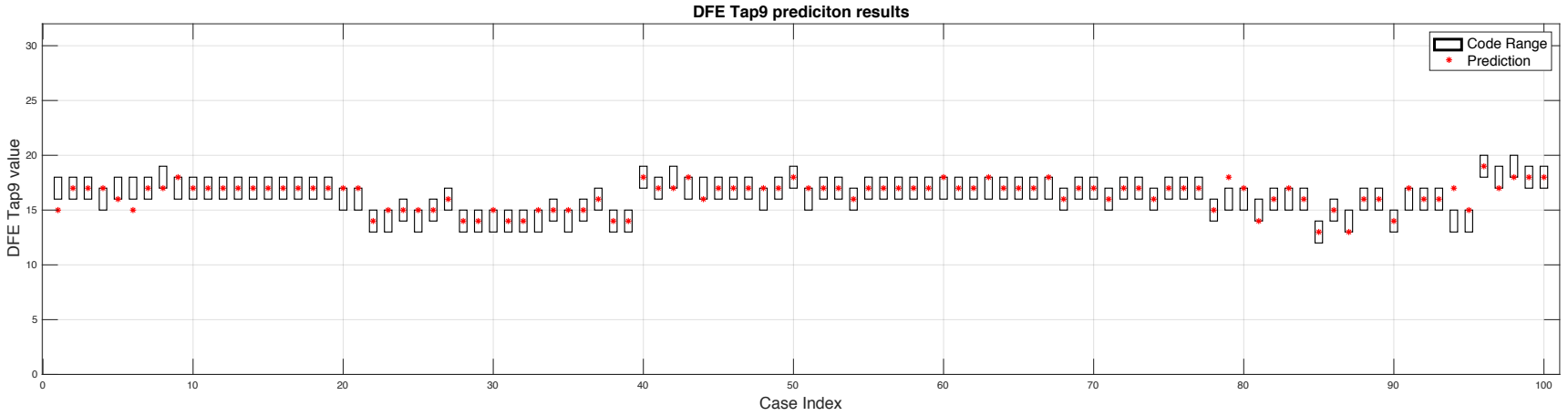
Some DFE tap prediction results in UltraScale+ GTY transceiver are shown below.

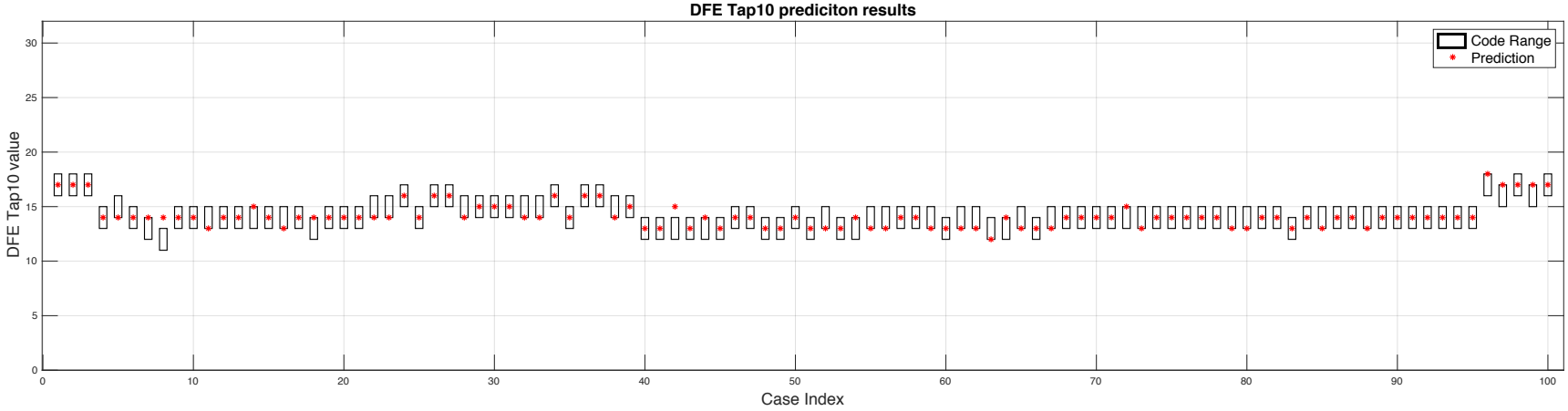


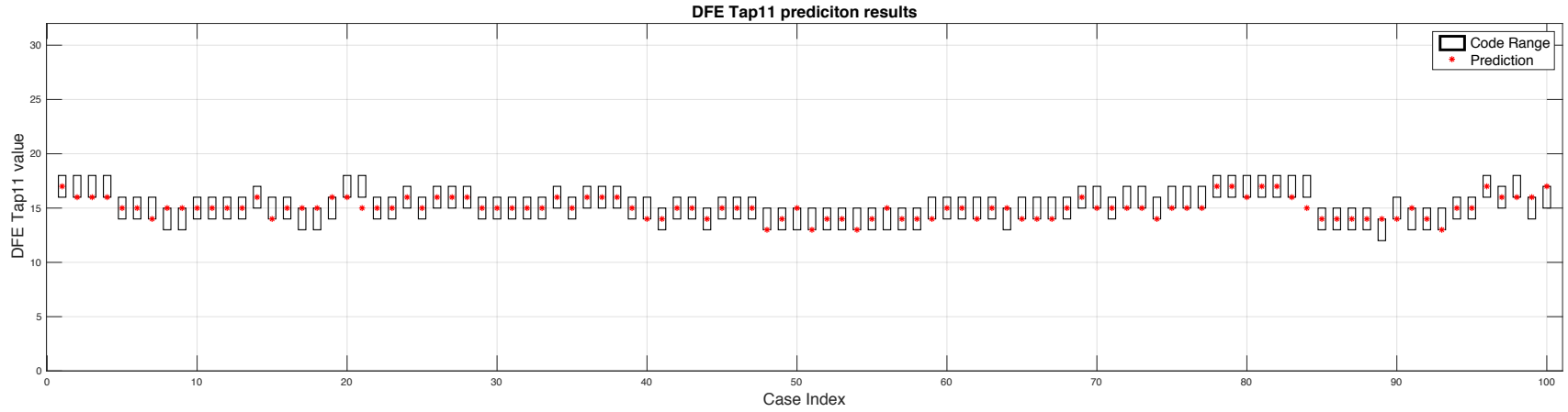


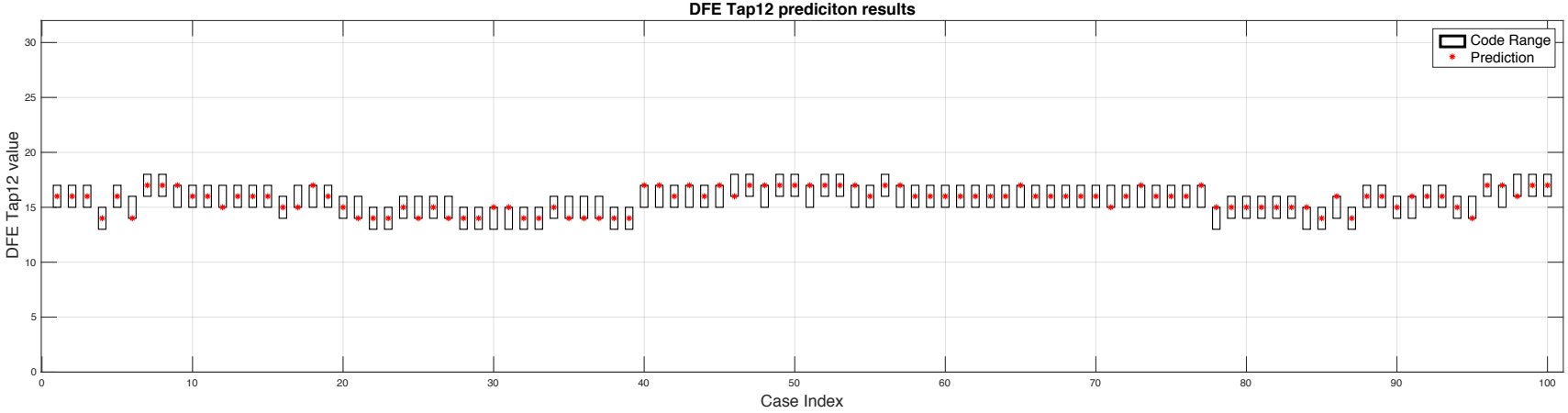

DFE Tap7 prediciton results
Code Range
Prediction
DFE Tap7 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100

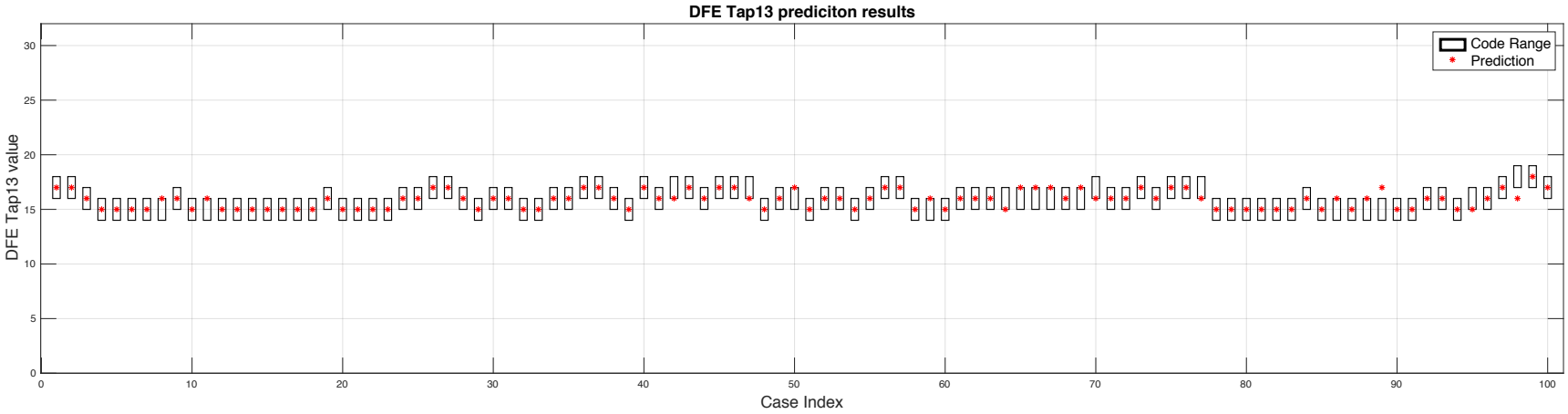

DFE Tap8 prediciton results
Code Range
Prediction
DFE Tap8 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100

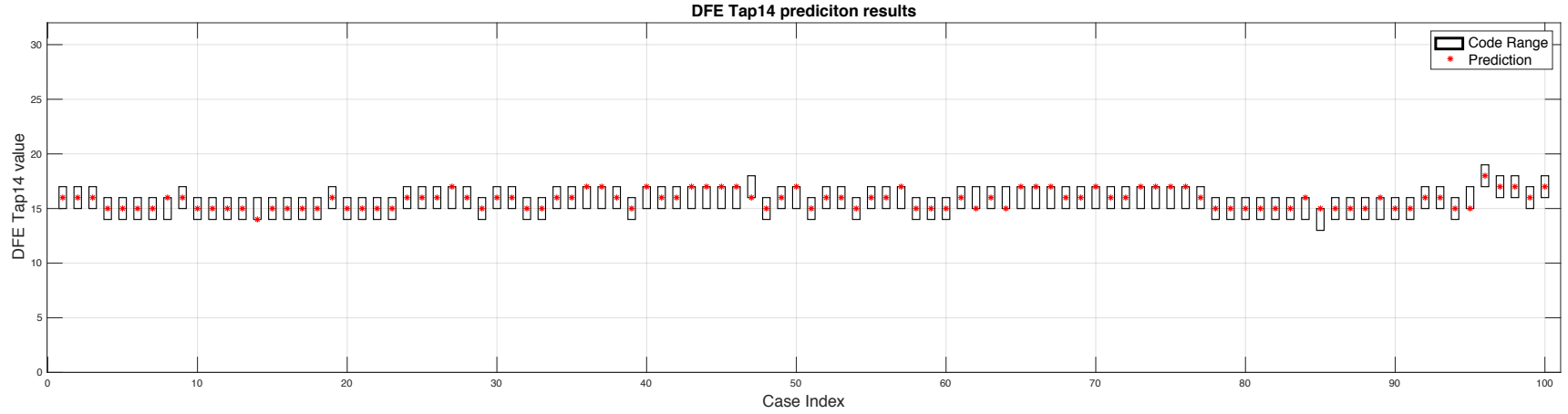

DFE Tap9 prediciton results
Code Range
Prediction
DFE Tap9 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100

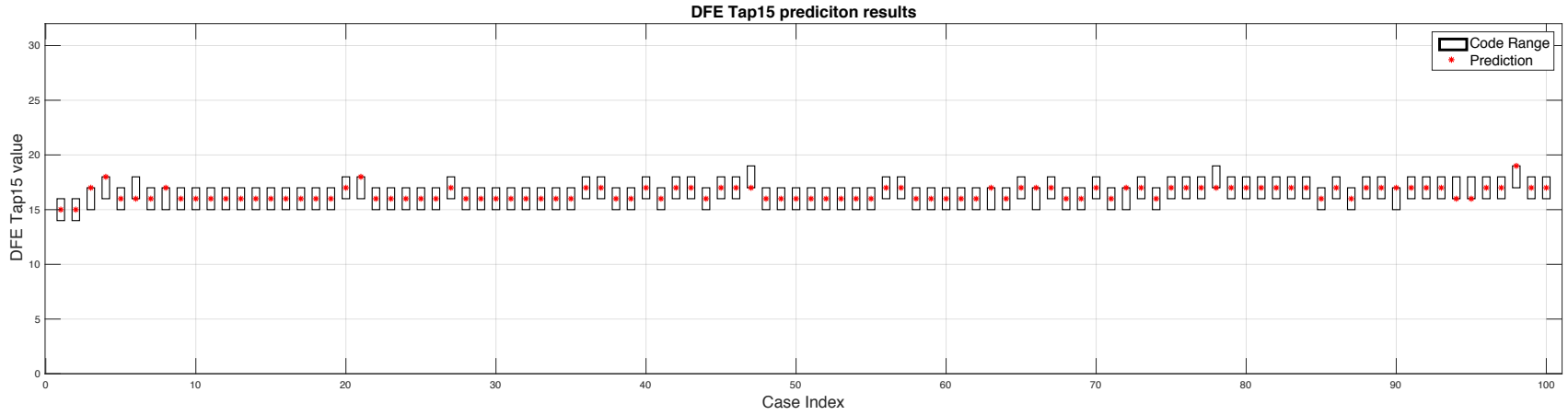

DFE Tap10 prediciton results
Code Range
Prediction
DFE Tap10 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100


DFE Tap11 prediciton results
Code Range
Prediction
DFE Tap11 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100


DFE Tap12 prediciton results
Code Range
Prediction
DFE Tap12 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100


DFE Tap13 prediciton results
Code Range
Prediction
DFE Tap13 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100


DFE Tap14 prediciton results
Code Range
Prediction
DFE Tap14 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100


DFE Tap15 prediciton results
Code Range
Prediction
DFE Tap15 value
Case Index
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
80
90
100