# Laguerre-Volterra Feed Forward Neural Network for Modeling PAM-4 High Speed Links

Xinying Wang, *Student Member, IEEE,* Thong Nguyen, *Student Member, IEEE,* and Jose Schutt-Aine, *Fellow, IEEE*

***Abstract*—In this paper, we present a PAM-4 IBIS-AMI model derived from machine learning for time domain simulation. More specifically, we report a Laguerre-Volterra-expanded feed-forward neural network (LVFFN) approach with one hidden layer and 10 neurons to model the 28Gbps PAM-4 high speed link buffer. The proposed LVFFN model reduces the model size and improves the computational efficiency dramatically compared to the Volterra series model and other transitional artificial neural network models. The LVFFN model is implemented in IBIS-AMI, an industrial standard, and is simulated in existing software platforms for eye-diagram analysis. This work has two innovations 1) We propose a method that dramatically reduces the neural network model complexity through a Laguerre-Volterra expansion when modeling weakly nonlinear systems with memory. (2) We implement a LVFFN model into IBIS-AMI to enhance the model interoperability and transportability.**

***Keywords*—PAM-4, Volterra Series, Neural Network, Behavior Modeling, Eye Diagram**

## I. Introduction

The growth of new technologies such as big data, cloud computing, the Internet-of-Things, 5G, and artificial intelligence, has driven demand for instant data exchange around the world. Global IP traffic will reach 4.8 zettabytes per year by 2022, a three-fold increase since 2017[1]. Boost in the data transmission rate to 400Gb/s and above has been called in the next generation high-speed link (HSL) technologies. Traditional non-return-to-zero (NRZ), also known as pulse-amplitude modulation 2-Level (PAM2), has been evolving over the past five decades from 10Gb/s to 100Gb/s (4 lanes, 25-28Gb/s/lane). Scaling up NRZ technology to 400Gb/s challenges the transceiver design for closed eye at receiver output. At 400Gb/s, NRZ experiences enhanced receiver sensitivity, tighter jitter budgets, and deteriorated signal-to-noise ratio. To open the eye, advanced correction techniques must be implemented at both transmitter and receiver side [2]. For instance, enhanced receiver equalization schemes such as continuous-time-linear equalization (CTLE) and decision-feedback-equalization (DFE) are needed to correct the channel loss and reflections.

To alleviate the challenges of linear data rate scale-up with NRZ technology, pulse amplitude modulation 4-level (PAM-4) is introduced in the IEEE P802.3bs standard [3]. Unlike NRZ, which has two levels (-1,1) encoding one bit (0, 1) for each level, PAM-4 has 4 amplitude levels, (-3, -1, 1,3) encoding 2 bits (00, 01, 10, 11) for each level. Therefore, one symbol in PAM-4 carries twice as much information as in NRZ, which can be interpreted as meaning that PAM-4 doubles the throughput for the same baud rate as NRZ. From the frequency domain perspective, PAM-4 requires only half of the bandwidth of NRZ for the same throughput. In light of these advantages, PAM-4 has gained growing attention since its introduction.

Although PAM-4 offers higher spectral efficiency, lower channel loss, and less stringent timing requirement, challenges lie in the design, test, and validation of PAM-4 transceivers [4]. Figure 1 illustrates the signal and eye diagram difference between NRZ and PAM-4. As seen, the time domain waveform for NRZ has two levels and only one eye is observed (Figure 1(a)) while that of PAM-4 has four levels and three eyes (Figure 1(b)). Thus, multi-level signal modulation in PAM-4 is the game-changer in transceiver design, test, and validation. New chip designs to support PAM-4 multi-level modulation will have to address the area and power increase to accommodate more transistors. The size of the integrated circuits (ICs) for PAM-4 has increased nearly 30%, while the power consumption increased up to 35% [4]. PAM-4 transceivers of 32Gb/s [5], 45Gb/s [6], and 56Gb/s [2] [7] fabricated with 90 nm, 60 nm, and 14 nm technologies have been reported. However, few papers are published on testing and validation. The reason is that the testing and validation strategy for the PAM-4 system does not share the same basis as that for the NRZ system due to multi-level signal modulation. Consequently, more sophisticated equipment is needed for PAM-4 system testing and validation. Moreover, due to the much higher data rate, the jitter budget for the PAM-4 system could be lower than the intrinsic jitter tolerance of the testing and validation equipment. Therefore, design and construction of such equipment are expensive and time-consuming.

The Department of Electrical and Computer Engineering,University of Illinois Urbana-Champaign, Champaign, IL, 61820 USA e-mail: xinying, tnnguye3, jesa@illinois.edu

Manuscript received XXXX, XXXX; revised XXXX, XXXX.

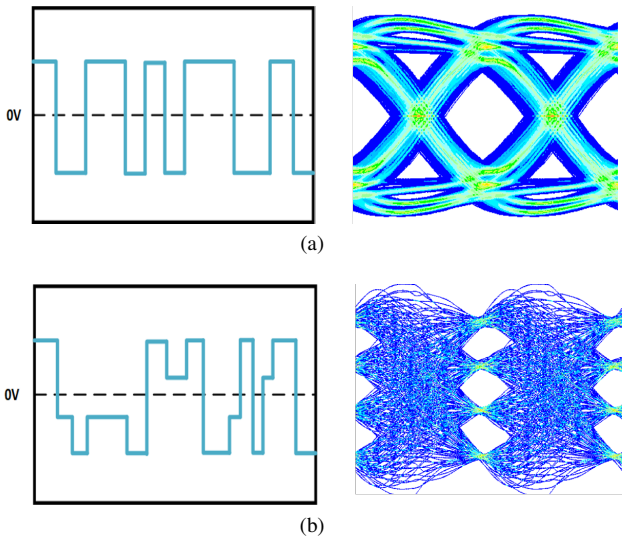



Fig. 1: Bit pattern and eye diagram for (a) NRZ and (b) PAM4

PAM-4 system simulation turns out to be a viable solution to alleviate the challenges off the PAM-4 system testing and validation. Compared to testing and validation of a PAM-4 system in a real production environment, verification of such a system through computer simulation can be fast and low cost. However, the success of this method heavily relies on the accuracy and efficiency of the PAM-4 behavioral model that is used in simulation. The behavioral model has two major advantages: (1) it is fast and computationally efficient, and (2) it is good for IP-protection. There are a variety of excellent behavioral models such as state-space [8], NARX [9], Verilog-A [10], Volterra series [11] [12], M/spl$\pi$/log [13]. Such models have achieved success in a various fields such digital IC, analog/mixed-signal module, RF amplifier/mixer, communication channels. However limited simulation work has been reported on PAM-4 behavior modeling.

In recent years, there has been growing interest in using machine learning methods for behavior modeling of high-speed links and integrated circuits. An increasing number of studies have demonstrated that multi-layer artificial neural networks and recurrent neural networks can be trained to accurately represent non-linear electrical circuits and systems, such as zero-in zero-out (ZIZO) circuits [15][10], PAM-4 system [16][17], and high-speed links [18]. In these models, a time-series input/output signal pair collected from either measurement of the real circuits or a SPICE model are used for machine learning model training. Once the models are trained, they can be used to model the real circuit/system for prediction. They are typically computationally faster than traditional models. Moreover, they are inherently suitable for IP protection. However, there are two main limitations associated with such models: they lack interoperability, and are computationally inefficient when more layers and more neurons are required. For instance, such models are not compatible with mainstream EDA software, which limits their application in industry. Therefore further improvement in such models is essential to promote their use in behavioral modeling of high-speed circuit/systems.

This paper aims to tackle the two aforementioned challenges associated with behavior models with machine learning. We propose a LVFFN which is able to drastically reduce the complexity of FFN otherwise employed. In addition, we implement this behavior model into IBIS-AMI, an industry standard, in order to enhance model interoperability and transportability. In other words, we develop a machine learning behavioral model that can be simulated using mainstream EDA circuit simulation software. This paper is organized as follows. LVFFN method is introduced in Section II. IBIS-AMI specification and implementation is introduced in Section III. In Section IV, the behavioral model of a PAM-4 system is demonstrated with LVFFN and benchmark comparison is carried out. IBIS-AMI model implementation and test in a simulator is demonstrated in Section V. Section VI concludes the paper and discusses future work.

## II. LVFFN

### A. Volterra Series

Volterra series is a versatile mathematical model describing nonlinear system with memory. Successful applications in modeling power amplifiers [19], [20], analog integrated circuits [21], image processing [22], coupled devices and circuits [23], nonlinear equalizers [24], and filters [25] have been reported. For discretized input-output data, the Volterra series of a causal, stable, nonlinear, time-invariant system is given by

$$\begin{aligned} y(n) =& h_0 + \sum_{m=0}^{M} h_1(m)u(n-m) \\ &+ \sum_{m_1=0}^{M}\sum_{m_2=0}^{M} h_2(m_1,m_2)u(n-m_1)u(n-m_2) \\ &+ \sum_{m_1=0}^{M}\sum_{m_2=0}^{M}\sum_{m_3=0}^{M} h_3(m_1,m_2,m_3)u(n-m_1) \\ &\times u(n-m_2)u(n-m_3) \\ &+ \ldots \end{aligned} \tag{1}$$

where $n$ denotes the time step, $u(n)$ the input data sequence, $y(n)$ the output data sequence, $M$ the system memory length and $h_i$ the $i^{th}$- order Volterra kernels (VKs). Equation (1) can be rewritten in the form

$$\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{h} \tag{2}$$

where $U$ denote the input Matrix with memory M and order N. $Y$ denotes the output vector:

$$\begin{aligned} &\boldsymbol{Y} = [y_1, y_2, ..., y_n]^T \\ &\boldsymbol{U} = [\boldsymbol{u_1}, \boldsymbol{u_2}, ..., \boldsymbol{u_n}]^T \\ &\boldsymbol{u_n} = [u_n(0), ..., u_n(m), u_n(0)u_n(0), u_n(0)u_n(1), ...] \\ &\boldsymbol{h} = [h_0, h_1(0), h_1(1), ..., h_1(m), h_2(0,0), h_2(0,1), ...] \end{aligned} \tag{3}$$

The VKs can be estimated with the least mean square (LMS) method:

$$\boldsymbol{h} = (\boldsymbol{U'U})^{-1}\boldsymbol{U'Y} \tag{4}$$

The number of kernel values needed to be estimated is a function of the memory length and the system order. The curse of dimensionality for Volterra series has greatly limited its applications. For example, to model a second order system with memory length of 50, Volterra series would need total $50 \times 50 + 50 = 2,550$ kernel values, and a fourth order system with the same memory length requires 6,377,550 kernel values. The number of kernel values increases exponentially with the system order and memory length. The total number of kernel values for a Nth order system with memory length M can be expressed as below:

$$\frac{(N+M)(N+M-1)...(M+1)}{N!} \tag{5}$$

At the certain point, the number of kernel values exceeds the number of data available for kernel extraction, which makes the problem intractable due to the vast number of parameters. This limitation leads to computational latency and over-fitting.

### *B. Laguerre-Volterra Expansion*

To address this challenge associated with Volterra series, one of the solution is to project the VKs on a set of orthonormal basis functions [26]. Laguerre functions are among the most popular ones. They are the solutions of Laguerre differential function. The general form of discrete Laguerre functions are shown below [27]:

$$\phi_r(\tau) = \alpha^{\frac{\tau-r}{2}}(1-\alpha)^{\frac{1}{2}}\sum_{k=0}^{r}(-1)^k\binom{\tau}{k}\binom{r}{k}\alpha^{r-k}(1-\alpha)^k \tag{6}$$

The VKs are expandable using Laguerre functions up to $R^{th}$ order

$$\begin{aligned}
h_0 &= \theta_0 \\
h_1(\tau) &= \sum_{r=1}^{r=R}\theta_r\phi_r(\tau) \\
h_2(\tau_1,\tau_2) &= \sum_{r_1=1}^{r_1=R}\sum_{r_2=1}^{r_2=R}\theta_{r_1,r_2}\phi_{r_1}(\tau_1)\phi_{r_2}(\tau_2) \\
h_n(\tau_1,...,\tau_n) &= \sum_{r_1=1}^{r_1=R}...\sum_{r_n=1}^{r_n=R}\theta_{r_1,...,r_n}\prod_{l=1}^{n}\phi_l(\tau_l)
\end{aligned} \tag{7}$$

TABLE I: Number of parameters required to model system with order up to $4^{th}$ and memory length up to 40 using Volterra Series

| | $1^{st}$ order | $2^{nd}$ order | $3^{rd}$ order | $4^{th}$ order |
|---|---|---|---|---|
| M = 10 | 10 | 110 | 1,110 | 11,110 |
| M = 20 | 20 | 420 | 8,420 | 168,420 |
| M = 30 | 30 | 930 | 27,930 | 837,930 |
| M = 40 | 40 | 1,640 | 641,640 | 3,201,640 |

where $\phi_r(\tau)$ denotes the discrete $r^{th}$ orthonormal basis function. Plugging equation (7) into equation (1):

$$\begin{aligned}
y(n) &= \theta_0 + \sum_{r=1}^{R}\theta_r\sum_{\tau=0}^{M}\phi_r(\tau)u(n-\tau) \\
&+ \sum_{r_1=1}^{r_1=R}\sum_{r_2=1}^{r_2=R}\theta_{r_1,r_2}\sum_{\tau_1=0}^{M}\phi_{r_1}(\tau_1)u(n-\tau_1)\sum_{\tau_2=0}^{M}\phi_{r_2}(\tau_2)u(n-\tau_2) \\
&+ ... \\
&+ \sum_{r_1=1}^{r_1=R}...\sum_{r_n=1}^{r_n=R}\theta_{r_1,...,r_n}\sum_{\tau_1=0}^{M}\phi_{r_1}(\tau_1)u(n-\tau_1)... \\
&... \sum_{\tau_n=0}^{M}\phi_{r_n}(\tau_n)u(n-\tau_n)
\end{aligned} \tag{8}$$

In equation (8), the term $\sum_{\tau=0}^{M}\phi_r(\tau)u(n-\tau)$ is nothing but the convolved output of the $r^{th}$ order Laguerre function and the time series input signal. Use symbol $\ell$:

$$\ell_r = \sum_{\tau=0}^{M}\phi_r(\tau)u(n-\tau) \tag{9}$$

Plugging equation (9) into equation (8):

$$\begin{aligned}
y(n) =& \theta_0 + \sum_{r=1}^{R}\theta_r\ell_r + \sum_{r_1=1}^{r_1=R}\sum_{r_2=1}^{r_2=R}\theta_{r_1,r_2}\ell_{r_1}\ell_{r_2} \\
&+ ... + \sum_{r_1=1}^{r_1=R}...\sum_{r_n=1}^{r_n=R}\theta_{r_1,...,r_n}\prod_{i=1}^{n}\ell_{r_i}
\end{aligned} \tag{10}$$

Equation (10) has the same form as equation (2). Parameter identification shifts from VKs $h_n$ in equation (2) to Laguerre parameters $\theta_r$ in equation (10). The number of parameters to be identified has been reduced dramatically. Typically the number of Laguerre functions $R$ employed for expansion is much smaller than the memory length $M$ which determines the number of kernel values. Table I and Table II illustrate the dimension reduction between Volterra series and Laguerre-Volterra expansion.

### *C. Parameter Identification with Neural Network*

Laguerre parameter identification is crucial. LMS has been widely used to obtain the Laguerre parameters. However, this method relies on matrix inversion and consequently becomes

TABLE II: Number of parameters required to model system with order up to $4^{th}$ and memory length up to 40 using Laguerre expansion

| | $1^{st}$ order | $2^{nd}$ order | $3^{rd}$ order | $4^{th}$ order |
|---|---|---|---|---|
| R = 2 | 2 | 6 | 14 | 30 |
| R = 3 | 3 | 12 | 39 | 120 |
| R = 4 | 4 | 20 | 84 | 340 |
| R = 5 | 5 | 30 | 155 | 780 |

unpractical when the parameter dimension becomes large. The other method is through feed-forward neural network. Geng et al. proposed a recurrent Laguerre-Volterra network which can identify the Laguerre parameters and even the decay factor $\alpha$ associated with Laguerre functions [28]. Such method has been successfully applied to model biological systems such as brain neuron cells. This method is specifically proposed for biological signals which have dynamics in the scale of several tens of Hz. However, it encounters difficulty when modeling an HSL system which usually runs at several tens of GHz or even hundreds of GHz. The latency depends largely on the channel length. Therefore, a novel parameter identification method is desirable for HSL systems.

In our previous works, we proposed a monomial power series neural network (MPSNN) to identify VKs for PAM-2 and PAM-4 HSL system [15][17]. The advantages of the proposed method are that it (1) allows the analytical mapping of trained weights to VKs, which produce higher accuracy and (2) reduces the number of parameters to be identified and enable reconstruction of the signal through extracted kernels. This method can be adapted to identify all the Laguerre parameters $\theta_r$ in equation (8). In the following, we will go through the method briefly.

Consider equation (2) where $n$ denotes the time step, $u(n)$ the input data sequence, $y(n)$ the output data sequence, $M$ the system memory length and $h_i$ the $i^{th}$ order VKs to be identified. This method employs a feed-forward neural network (FFN) with one input layer, one hidden layer, and one output layer. The output layer has a single output node. The number of neurons in the hidden layer takes the system memory length. The mathematical representation is given by:

$$y(n) = \sum_{i=1}^{M} c_i \sigma_i \left( b_i + \sum_{j=0}^{M} w_{ij} u(n-j) \right) \tag{11}$$

where $y(n)$ is the output, $u(n)$ the input, $c_i$ the weight from hidden unit $i$ to the output, $w_{ij}$ the weight from input $i$ to hidden unit $j$, and $\sigma_i$ the activation function. The activation function, if infinitely differentiable, can be represented by an equivalent truncated polynomial function $g_i$ given by

$$g_i(x) = \sum_{j=0}^{M} a_{ji} x^j \tag{12}$$

where $x^j$ is given by

$$x^j = u(n)^{v_0} u(n-1)^{v_1} ... u(n-M)^{v_M}|_{v_0+v_1+...+v_M=j} \tag{13}$$

Combining Equations (12) and (13) and plugging into (11), we can represent the output $y(n)$ in the form

$$\begin{aligned} y(n) = &\sum_{i=1}^{M} c_i a_{0i} + \sum_{i=1}^{M} c_i a_{1i} w_{ij} u(n-j) + ... \\ &+ \sum_{i=1}^{M} c_i a_{ni} w_{v_1 i} w_{v_2 i} ... w_{v_n i} \\ &\times u(n-v_{1i}) u(n-v_{2i}) ... u(n-v_{ni}) \end{aligned} \tag{14}$$

Comparing (14) to (2), the Volterra kernel can be represented by

$$h_0 = \sum_{i=1}^{M} c_i a_{0i} \tag{15}$$

$$h_1(j) = \sum_{i=1}^{M} c_i a_{1i} w_{ji} \tag{16}$$

$$h_2(j,k) = \sum_{i=1}^{M} c_i a_{2i} w_{ji} w_{ki} \tag{17}$$

$$h_n(v_1, v_2, ..., v_n) = \sum_{i=1}^{M} c_i a_{ni} w_{v_1 i} w_{v_2 i} ... w_{v_n i} \tag{18}$$

where $h_n$ is the $n^{th}$ VK, $c_i$ and $w_{ni}$ the weights obtained from the FFN (see Figure 2(a)), and $a_{ni}$ are the parameters in equation (12). Obtaining the correct $a_{ni}$ is crucial for accurately mapping the FFN weights to the Volterra kernels. One popular method uses an hyperbolic tangent activation function, $\sigma(x) = \tanh(x)$. The $a_{ni}$ can be obtained from the Taylor expansion of $\tanh(x)$ [29]. However, this method suffers from large number of kernels, inaccurate kernel estimation, and difficulty in signal reconstruction with estimated kernels. In our method, the coefficients $a_{ni}$ are obtained by direct expansion of a monomial power series activation function. Such activation function is given by

$$\sigma(x) = x^n \tag{19}$$

The nodal output function can be obtained by replacing $x$ with (13) and given by

$$\sigma(x) = \left( b_i + \sum_{j=0}^{M} w_{ij} u(n-j) \right)^n \tag{20}$$

To simplify the derivation, we can represent $w_{ij}u(n-j)$ with $u_j$ and $b_i$ with $b$. The simplified equation (20) is given by

$$\sigma(x) = (b + u_0 + u_1 + ... u_M)^n \tag{21}$$

Direct expansion of (21) is straightforward. The coefficient $a_{ni}$ can be obtained from the $n^{th}-$ power term in the expansion, which is given by

$$a_{ni} = b_i^{k_b} \begin{pmatrix} n \\ k_0, k_1, k_2, ..., k_M \end{pmatrix}_{k_b+k_0+....+k_M=n} \tag{22}$$

where $n$ is the order of the kernel, $k_b$ the order of $b$ in the expansion of (21), and $k_0, k_1, ..., k_M$ are the order of

$u_0, u_1, ..., u_M$, respectively. The sum of all parameters $k$ has to be equal to the order $n$

The proposed method employs the same feed-forward neural network structure in which only one hidden layer is used. Promisingly, it can be adapted to allow multiple hidden layers to achieve higher order Volterra kernel identification. Since there is no truncation needed in the activation function expansion, our method produces more accurate kernels. The number of parameters will be regulated by the order of the activation function, so the total number of parameters to be identified is reduced significantly.

Identification of Laguerre parameters is the same as Volterra kernel identification except that an extra layer is added between the input layer and the hidden layer to compute the convolution of the input with the Laguerre functions. The neural network used for identification of Laguerre parameters is illustrated in Figure 2(b). The added layer which is labeled in the red frame does not change the derivation above for parameter identification. The Laguerre parameter $\theta_r$ takes the general form:

$$\theta_{r_1,r_2,...,r_n} = \sum_{i=1}^{R} c_i a_{ni} w_{r_1 i} w_{r_2 i} ... w_{r_n i} \tag{23}$$

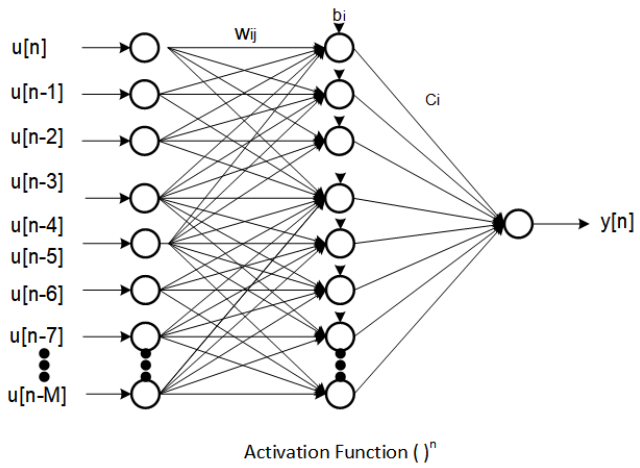
where $a_{ni}$ is calculated through equation (22); $R$ denotes the number of Laguerre functions employed for expansion. In this work, we choose the third order monomial power series, $\sigma = x^3$, as activation function. Laguerre parameters up to the third order can be calculated through equation (24) to (27):

$$\theta_0 = \sum_{i=1}^{R} c_i b_i^3 \tag{24}$$

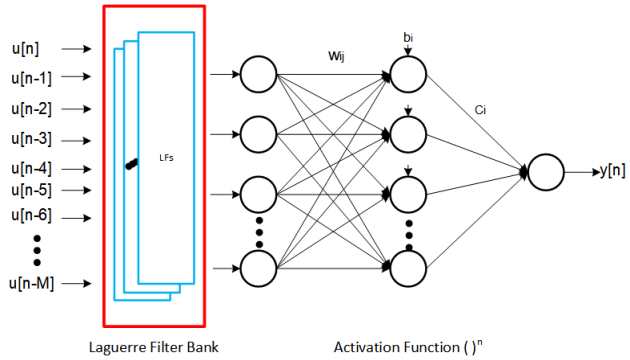
$$\theta_{r_1} = \sum_{i=1}^{R} c_i b_i^2 w_{r_1 i} \tag{25}$$

$$\theta_{r_1,r_2} = \sum_{i=1}^{R} c_i b_i w_{r_1 i} w_{r_2 i} \tag{26}$$

$$\theta_{r_1,r_2,r_3} = \sum_{i=1}^{R} c_i w_{r_1 i} w_{r_2 i} w_{r_3 i} \tag{27}$$




(a)




(b)

Fig. 2: Neural network architectures for (a) FFN and (b) LVFFN

## III. IBIS-AMI Model Basics

The IBIS-AMI standard specifies the interface between a behavioral model and a simulator. A typical IBIS-AMI model contains analog and algorithmic portions. The analog component refers to the IBIS part in which the package and channel response are presented. The algorithmic part contains an executable file and a .ami file in which all the parameters used by the executable and simulator are specified.

Figure 3 illustrates the composition of the IBIS-AMI model. The analog part is composed of a list of S-parameter files which describe the channel and package electrical behavior. They are SNP files for the simulator to generate impulse response. The algorithmic model, on the other hand, has two parts, the transmitter (Tx) and the receiver (Rx). Both are implemented into separate dynamic link library (DLL) files using C++. The Tx portion pre-processes the data stream if there is a feed-forward equalizer (FFE) present. It either modifies the impulse response or returns the data bits. In the Rx portion, the Rx DLL implements a post-processing block, the decision feed back equalizer (DFE), and a clock-data recovery (CDR) block. The output of the Rx DLL is either the recovered data bits and clock data or the modified impulse response which is then convolved with the data stream

returned by the Tx DLL.

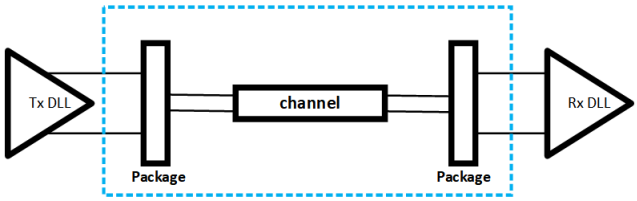



Fig. 3: IBIS-AMI model illustration

The IBIS-AMI specification defines the application programing interface (API) functions which provide a platform for the user to implement a customized algorithmic model for SerDes simulation. Those API functions include three main functions: *AMI_Init()*, *AMI_GetWave()*, and *AMI_Close()*. *AMI_Init()*, as its name implies, mainly conducts initialization which includes allocating/initializing the dynamic memory, parsing parameters from .ami file, and conducting impulse response filtering or statistical simulation for LTI system. *AMI_GetWave()*, on the other hand, conducts bit-to-bit simulation for systems that non-linear and time-variant. *AMI_GetWave()* is an optional function which can be explicitly disabled in the .ami file. If this function is disabled, the impulse response returned by *AMI_Init()* has to be enabled, and vice versa. However, enabling/disabling both at the same time is not permitted. The last function, *AMI_Close()*, typically conducts post-simulation concluding and cleaning, which include stopping the simulation engine, releasing the allocated memory, and informing the simulator that the simulation has been completed successfully or with errors.

The whole IBIS-AMI simulation process starts with generation of the PRBS excitation bit sequence, parsing the .ami file, and obtaining the impose response from S-parameter files. Subsequently, the simulator presents all the information to the Tx AMI model. Depending on the user's specification, whether *AMI_GetWave()* is disabled or not, the simulator will behave differently. If no *AMI_GetWave()* is available, the impose response returned by *AMI_Init()* is convolved with the initial excitation bit sequence. Otherwise, the bit sequence returned by *AMI_GetWave()* is then convolved with the analog impose response to generate the data sequence for the following processing. Next, the resulting waveform is presented to Rx AMI model. The Rx DLL behaves similarly to Tx DLL. Depending on *AMI_GetWave()* enabled or not, it either only modifies impose response or conducts bit-to-bit processing. The resulted output is either impose response from Rx *AMI_Init()* or the waveform from Rx *AMI_GetWave()*. The simulator will then take the output returned by Rx DLL to conduct the final eye diagram analysis.

IBIS-AMI model is an industry standard model which is usually supplied by SerDes vendors. Generating an IBIS-AMI model is not trivial, requiring knowledge in electronic circuit design, signal integrity, and software development. There are a few tools available facilitating IBIS-AMI model generation; however they lack flexibility for accommodating novel high-speed link behavior model development. In this work, we propose an approach to implement the PAM-4 LVFFN behavior model into IBIS-AMI. This model can be simulated in mainstream EDA circuit simulation softwares.

## IV. Modeling PAM-4 System with LVFFN

In this section, a PAM-4 high-speed link system is modeled with LVFFN. A block diagram of a typical high speed link system is illustrated in Figure 4. The system consists of three portions: transmitter, channel, and receiver. To compensate for the signal loss and distortion during transmission, correction blocks such as equalizers, feed-forward equalizer (FFE), and decision-feedback equalizer (DFE) are employed. Multi-level signaling presented in the PAM-4 system severely complicates the model development. To demonstrate the effectiveness of LVFFN on modeling the PAM-4 system, we model the system with both a regular FFN and an LVFFN proposed in this paper.

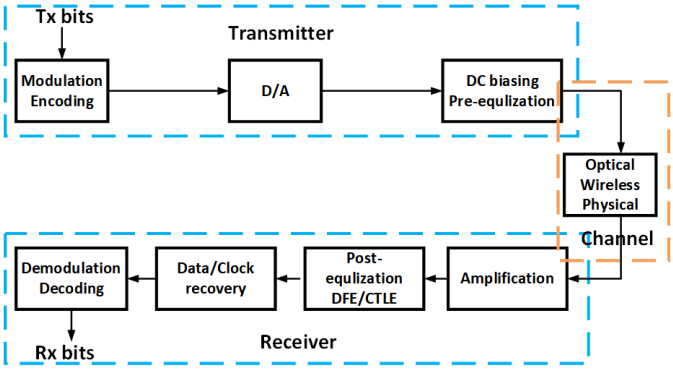



Fig. 4: PAM-4 high speed link system block diagram

Figure 5 shows the input and output signals that are used for training the models. The input/output signals are generated from an industrial PAM-4 IBIS-AMI model. The input signal is a 4-level pseudo-random binary sequence (PRBS) with amplitude between -1.0 V and 1.0 V. The data rate is 28Gb/s. Total 16,651 time series samples are collected for training and testing (70/30%). The regular FFN model employed in this work has three layers: one input, hidden, and output. A third-order monomial power series is used for the activation function. Determining the right memory length for the system under modeling is important. In our previous work, we have demonstrated that longer memory length would result in better accuracy. There is an elbow point for the memory length after which increasing length will not obviously benefit accuracy. We found that for PAM-4 system under modeling, memory length of 150 produces the optimal result. In this work, we use the memory length of 150 for both FFN model and LVFFN model. Back-propagation algorithm with an Adam optimizer is selected for minimizing the Mean Square Error (MSE). Dropout is not used during training. The learning rate typically takes value in between $10^{-3}$ to $10^{-6}$. Bigger learning rate leads to faster converge

but worse training/testing accuracy. Typically the training accuracy get above 90% in less than 25 epochs or less than 2 minutes CPU time using a desktop equipped with an Intel i7 quad core CPU. The best training/testing accuracy of 98%/97% can be achieved by allowing a longer training time.

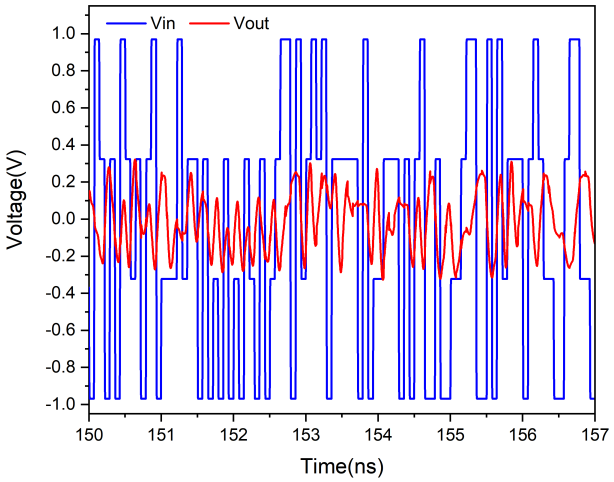



Fig. 5: Plot of PAM-4 input output data

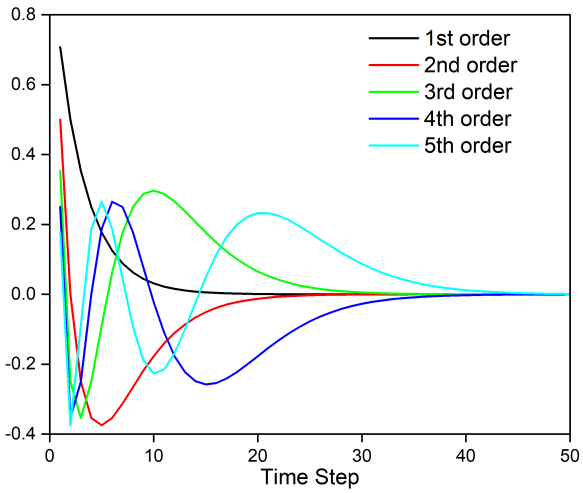



Fig. 6: Plot of first five Laguerre functions

### *A. Determining Decay factor $\alpha$ and the Order Laguerre Function*

In LVFFN, the input signal is first convolved with the Laguerre functions. Figure 6 plots the first five Laguerre functions for $\alpha$ = 0.2 and $\alpha$ = 0.5. As we can see, the decay factor controls how fast the Laguerre function approaches zero. Finding the right decay factor $\alpha$ is crucial for modeling. There have been works reporting on how to obtain optimal $\alpha$ for a system. Campello et al. reported that in $Z$ domain, the corresponding $r^{th}$ Laguerre functions are given by [30]

$$\Phi_r(z) = \frac{z\sqrt{1-P_r^2}}{z-P_r}\left(\frac{1-P_r z}{z-P_r}\right)^n \tag{28}$$

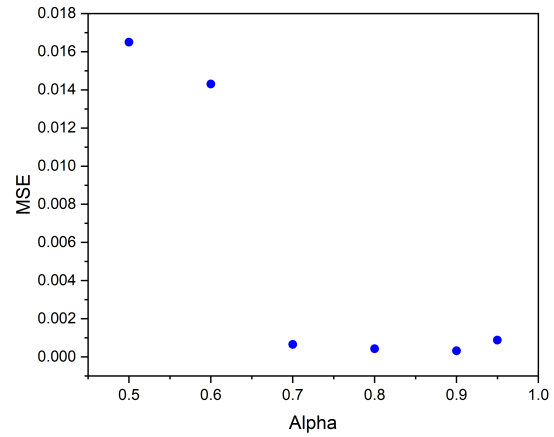
$P_r$ is also called a real number Laguerre pole. Then the optimal Laguerre pole $P_r$ can be obtained by solving the following optimization problem:

$$\min_{-1<p_r<1} J_r = \sum_{i_1=1}^{\infty} \dots \sum_{i_r=1}^{\infty} (i_1 + \dots + i_r)\alpha_{i_1,\dots,i_r}^n \tag{29}$$

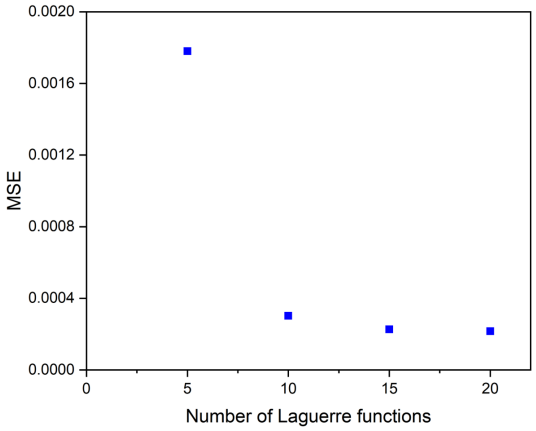
The optimal pole can be obtained using equation (29). However the process could be computationally prohibitive. Another approach to obtain the right $\alpha$ takes advantage of neural network training. $\alpha$ is treated as one of the parameters in the neural network and then converges to the optimal value through back-propagation.

In the expansion, the number of Laguerre functions is truncated to a certain quantity. Determining the minimum number of functions that are enough to achieve the desired model accuracy is another challenge. Typically, more functions are desired to achieve better prediction accuracy. However, more functions will deteriorate the benefit of Laguerre expansion for dimension reduction. Kang and Marmarelis et al. reported a method of principal dynamic mode analysis to obtain the least number of Laguerre functions. This method has also been applied to model EEG data for diagnosis of Alzheimer's disease [31]. In this method, the first and second order VKs have to be identified using a traditional algorithm such as LMS. Then singular value decomposition (SVD) is applied to obtain the most significant elements on the first and second order VK matrix. The number of significant VK elements relates to the number of Laguerre functions is used in the expansion.

In this work, a simple brute force algorithm is used to obtain the optimal $\alpha$ and the least number of Laguerre functions. Figure Figure 7 shows the plot for $\alpha$ value ( Figure 7 (a)) and number of Laguerre functions (Figure 7 (b)) versus MSE value. As seen in the figure, MSE magnitude approaches equilibrium as both $\alpha$ value and the number of functions increase. Based on the plot, we can determine the optimal value of $\alpha$ and the number of Laguerre functions for the PAM-4 system. The optimal value of $\alpha$ is determined to be 0.91, and the number of Laguerre functions is selected to be 10.




(a)




(b)

Fig. 7: Plots of MSE vs (a) $\alpha$, and (b) number of Laguerre functions

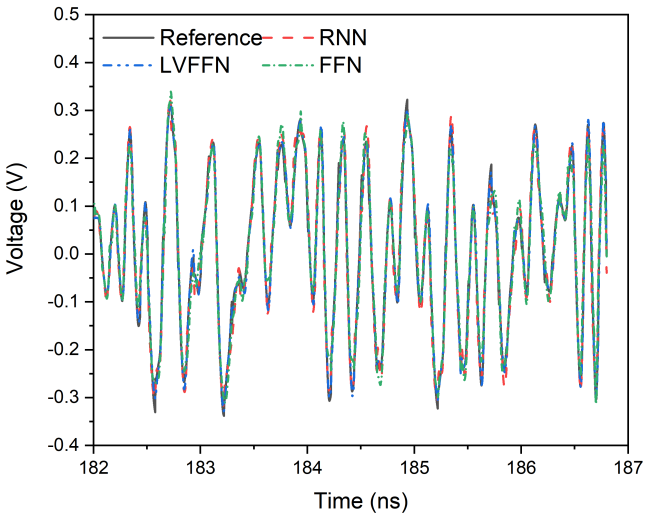



Fig. 8: Model output comparison of LVFFN, RNN, FFN, and the reference signal

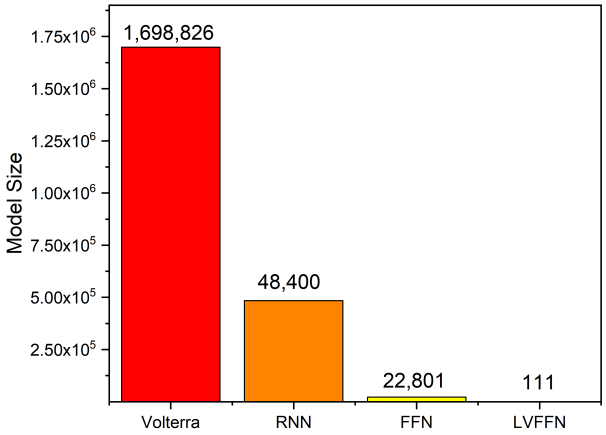



Fig. 9: Comparison of model size for Volterra series, RNN, FFN, and LVFFN

### *B. Modeling Result*

The PAM-4 system is modeled with LVFFN, regular FFN, and recurrent neural network (RNN). In the RNN model, 6 stacked layers with 20 neurons for each layer and memory length of 100 are employed [16]. Figure 8 plots the model output of LVFFN, FFN, RNN, and reference signal. As seen, all the models demonstrate great accuracy. The FFN model needs 1 hidden layer and 150 neurons, while LVFFN only needs 1 hidden layer and 10 neurons.

Model size comparison for Volterra series, RNN, FFN, and LVFFN is presented in Figure 9. From the regular FFN network, VKs up to the $3^{rd}$ order can be obtained through equation (15)-(18). With the identified VKs, the PAM-4 system can be completely described. However, according to equation (5), we can calculate the total number of kernel value for the system of $3^{rd}$ order non-linearity and memory length of 150. The number of kernel value is 3,397,651. VKs have the attribute of symmetry, e.g. the $h(a, b)$ is identical to $h(b, a)$. Taking this attribute into account, the total number of kernel value is reduced to 1,698,826, which is still an exceptionally large model. The model size that comes in the second place is the RNN model which has 485,200 parameters. Following RNN, the regular FFN requires 22,801

parameters. The LVFFN only needs 111 parameters which is substantially smaller for similar prediction accuracy. This is a significant model size reduction compared to other regular machine learning models and Volterra series. The model size reduction will improve the model transportability and computation efficiency.

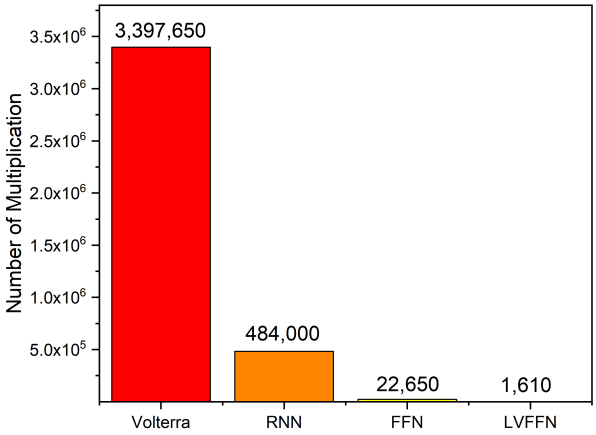



Fig. 10: Comparison of number of multiplications for calculation of one output sample for Volterra series, RNN, FFN, and LVFFN

Figure 10 shows the computational efficiency comparison for computing one output sample. The number of multiplications is used here as the criterion to characterize the model performance. As seen, LVFFN only requires 1.61K multiplications to compute one output sample. The total number of multiplications for computing one output sample includes the computation of the neural network, which is only 110, and the convolution of the input signal with the 10 Laguerre functions, which is 1500. As we can see, the improvement in computation efficiency is significant for the proposed LVFFN model. In LVFFN, the majority of the computational power is used by the convolution of input signal with each Laguerre function. This computation can be further reduced by calculating the filtered output recursively using [28]:

$$l_r(n) = \sqrt{\alpha} l_r(n-1) + \sqrt{\alpha} l_{r-1}(n) - l_{r-1}(n-1) \quad (30)$$

Initialization is carried out by

$$l_0(n) = \sqrt{\alpha} l_0(n-1) + \sqrt{1-\alpha} x(n) \quad (31)$$

System identification is popular in modeling non-linear systems. Typically, VKs up to the desired order are identified with time series input signal. In our previous work, we have reported MPSNN to extract VKs up to the third order [15] for PAM-4 HSL system. In this work, we are able to extract the VKs up to the third order using LVFFN for the same HSL system. In LVFFN, we can identify Laguerre parameters. Then the Volterra kernels up to desired order can be obtained through extracted Laguerre parameters using equation (8)

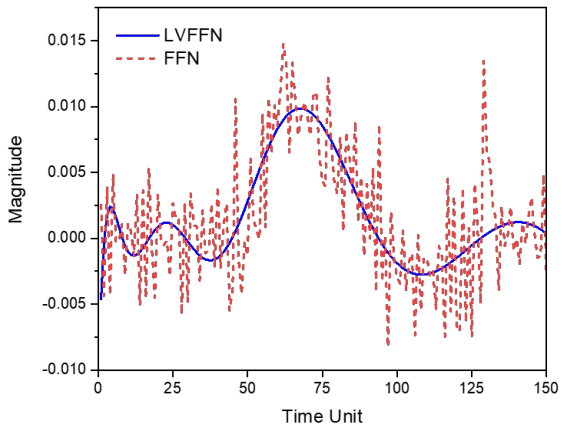



Fig. 11: First order VK extracted with FFN and LVFFN

Figure 11 shows the first order VKs extracted for PAM-4 system using FFN and LVFFN. As we can see, the VKs extracted from both methods matche fairly well. The kernel identified with LVFFN is smoother than the ones with FFN. This is expected as the Laguerre expansion of VK is equivalent to orthonormal decomposition of VK, which removes redundant information during the expansion process.

## V. IBIS-AMI Implementation

The trained LVFFN is implemented in IBIS-AMI DLLs. Figure 12 presents the simulation environment in ADS for the LVFFN IBIS-AMI model. The Tx DLL is a do-nothing model which passes the bits received from simulator to Rx DLL. The whole PAM-4 LVFFN model is implemented in Rx DLL. There is no channel between the Tx DLL and the Rx DLL since the LVFFN model incorporates the whole PAM-4 system including channel and packages. The eye diagram is plotted on the output from the LVFFN model for signal integrity analysis.

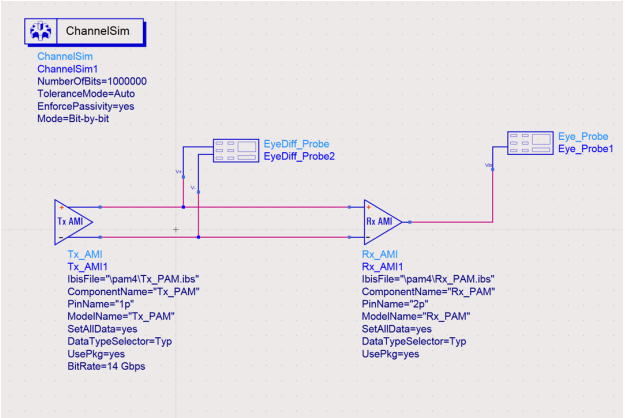



Fig. 12: Snapshot of simulation environment in ADS for LVFFN IBIS-AMI model

In the Rx DLL, three main functions are implemented by user: *AMI_init()*, *AMI_getWave()*, and *AMI_close()*. The

execution flow of the Rx DLL is illustrated in Figure 13. Once the bit stream is presented at the input of Rx DLL, the simulator calls the Rx *AMI_init()* function in which necessary memory is allocated and the Laguerre functions and the trained FFN parameters are loaded. At the end of the execution of the *AMI_init()* function call, messages containing information such as execution status, debug information, intermediate values, and so on are printed at the simulator interface. After the *AMI_init()* is returned, the simulator will then call the *AMI_getWave()* function multiple times to conduct the bit-to-bit simulation. The bit information is conveyed to the LVFFN implemented in this function and then return the value processed back to the simulator. Once all the bits are processed, the simulator will call the *AMI_close()* function to clean up the memory allocated and inform the simulator that the whole simulation has been completed successfully. Subsequently, the simulator will launch an eye diagram analysis routine to conduct the signal integrity analysis.

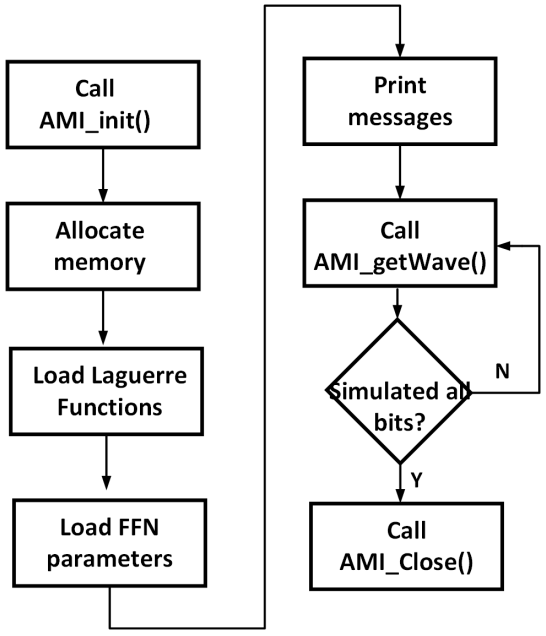
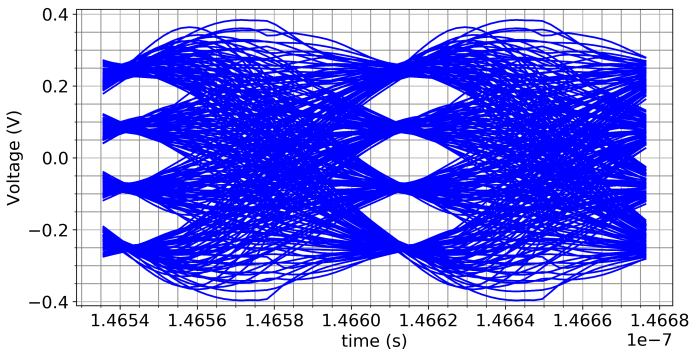
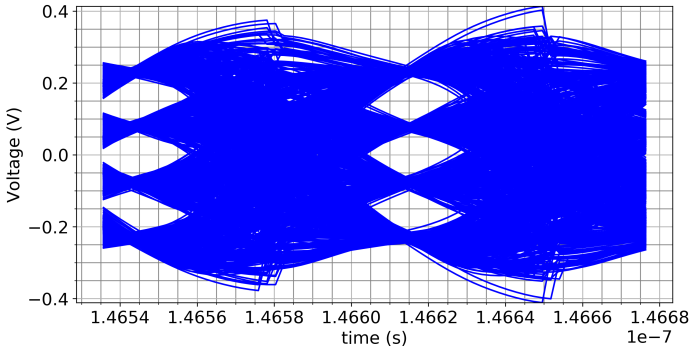



Fig. 13: Flow diagram of Rx DLL execution

Figure 14 shows the eye-diagram plots for both the PAM-4 LVFFN model and the reference model. The bits for the LVFFN model are generated from ADS and the reference signal is from an industrial IBIS-AMI model. There are three eyes in the eye-diagram plot for the PAM-4 model due to the multi-level signaling. As seen in the figure, eyes in both plots look very similar. Simulation of 1 million bits in ADS with the LVFFN model takes 142s. The majority of computational power is used by the convolution of Laguerre functions with the input time series signals. Further speed-up could be achieved by replacing convolution with recursive computation of the output using equations (30) and (31). In addition, the convolution portion can be easily accelerated by parallel computing. An FPGA-based hardware accelerator can be implemented to further enhance the computational speed.




(a)




(b)

Fig. 14: Eye-diagram plot for (a) PAM-4 LVFFN model and (b) reference model

## VI. Conclusion

In this work, we proposed an LVFFN approach to model the PAM-4 HSL system. In our approach, the time series input is preprocessed by convolving the input with a certain number of orthonormal Laguerre functions. The convolved output is then fed to the FFN for training and prediction. The LVFFN model with only one hidden layer and 10 neurons can be used to model the whole PAM-4 system. The model size is dramatically reduced due to the signal preprocessing. Compared to other models such as FFN, RNN, and Volterra series, the model size for LVFFN is reduced by 5 orders of magnitude for the same prediction accuracy. The improvement in computation efficiency for LVFFN is significant as well. To compute one sample at output, the LVFFN only requires 1.61K multiplications, while regular FFN, RNN, and Volterra series would need 22.65K, 485.2K, and 3,397.65K multiplications, respectively.

In addition, LVFFN can be used to conduct non-linear system identification. The first three order VKs are identified with the LVFFN. The highest order VKs are controlled by the activation function. We compared the first order VK identified with LVFFN to the one identified with regular FFN. They match very well and VK identified with LVFFN is more accurate.

The behavioral model generated from machine learning cannot be directly simulated in circuit simulator software,

which significantly limits its application in practice. We successfully implemented the LVFFN model in IBIS-AMI, a industrial standard. The model was verified using ADS from Keysight. Eye diagram analysis were conducted in ADS as well. The eye diagrams from LVFFN obtained with ADS appears very similar to those generated with the reference model.

## ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. CNS 16-24810 for the Center for Advanced Electronics through Machine Learning (CAEML), the U.S Army Small Business Innovation Research (SBIR) Program office and the U.S. Army Research Office under Contract No.W911NF-16-C-0125 and by Zhejiang University under grant ZJU Research 083650.

## REFERENCES

[1] "Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet." [Online]. Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935

[2] J. Lee, P.-C. Chiang, P.-J. Peng, L.-Y. Chen, and C.-C. Weng, "Design of 56 gb/s nrz and pam-4 serdes transceivers in cmos technologies," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 9, pp. 2061–2073, September 2015.

[3] "Ieee P 802.3bs Baseline Summary," 2016. [Online]. Available: http://www.ieee802.org/3/bs/

[4] "PAM-4 Design Challenges and the Implications on Test ." [Online]. Available: http://literature.cdn.keysight.com/litweb/pdf/5992-0527EN.pdf

[5] O. Elhadidy, A. Roshan-Zamir, H.-W. Yang, , and S. Palermo, "A 32 gb/s 0.55 mw/gbps pam4 1-fir 2-iir tap dfe receiver in 65-nm cmos," in *2015 Symposium on VLSI Circuits Digest of Technical Papers*, 2015, pp. C224–C225.

[6] M. Bassi, F. Radice, M. Bruccoleri, S. Erba, and A. Mazzanti, "A 45gb/s pam-4 transmitter delivering 1.3vppd output swing with 1v supply in 28nm cmos fdsoi," in *2016 IEEE International Solid-State Circuits Conference*, 2016, pp. 66–67.

[7] A. Roshan-Zamir, T. Iwai, Y.-H. Fan, A. Kumar, H.-W. Yang, L. Sledjeski, J. Hamilton, S. Chandramouli, A. Aude, and S. Palermo, "A 56 gb/s pam4 receiver with low-overhead threshold and edge-based dfe fir and iir-tap adaptation in 65nm cmos," in *Custom Integrated Circuits Conference*, 2018, pp. 1–4.

[8] I. S. Stievano, C. Siviero, F. G. Canavero, and I. A. Maio, "Behavioral modeling of digital devices via composite local linear state-space relations," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 8, pp. 1757–1765, 2008.

[9] I. S. Stievano, I. A. Maio, and F. G. Canavero, "Behavioral models of i/o ports from measured transient waveforms," *IEEE Transactions on Instrumentation and Measurement*, vol. 51, no. 6, pp. 1266–1270, 2002.

[10] Z. Chen, M. Raginsky, and E. Rosenbaum, "Verilog-a compatible recurrent neural network model for transient circuit simulation," pp. 1–3, 2017.

[11] J. Dooley, B. O. Brien, and T. J. Brazil, "Behavioural modelling of rf power amplifiers using modified volterra series in the time domain," in *High Frequency Postgraduate Student Colloquium*, Sep. 2004, pp. 169–174.

[12] C. Crespo-Cadenas, J. Reina-Tosina, and M. J. Madero-Ayora, "Volterra behavioral model for wideband rf amplifiers," in *IEEE Transactions on Microwave Theory and Techniques*, vol. 5, no. 3, 2007, pp. 449–457.

[13] I. S. Stievano, I. A. Maio, and F. G. Canavero, "M$\pi$log macromodeling via parametric identification of logic gates," *IEEE Transactions on Advanced Packaging*, vol. 27, pp. 15–23, 2004.

[14] P. Manfredi, D. V. Ginste, I. S. Stievano, D. D. Zutter, and F. G. Canavero, "Stochastic transmission line analysis via polynomial chaos methods: An overview," pp. 77–84, 2017.

[15] X. Wang, T. Nguyen, and J. Schutt-Aine, "Volterra kernel extraction through monomial power series feed forward neural network for behavior modeling of high speed i/o buffer," in *EMC Sapporo & APEMC 2019*, 2019, pp. 1–4.

[16] T. Nguyen, T. Lu, K. Wu, and J. Schutt-Aine, "Fast Transient Simulation of High-Speed Channels Using Recurrent Neural Network," 2019. [Online]. Available: https://arxiv.org/pdf/1902.02627.pdf

[17] T. Nguyen, X. Wang, X. Chen, and J. Schutt-Aine, "A deep learning approach for volterra kernel extraction for time domain simulation of weakly nonlinear circuits," in *2019 IEEE 69th Electronic Components and Technology Conference (ECTC)*, 2019, pp. 1889–1896.

[18] R. Trinchero, P. Manfredi, I. S. Stievano, and F. G. Canavero, "Machine learning for the performance assessment of high-speed links," *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, pp. 1627–1634, 2018.

[19] J. Dooley, B. O. Brien, and T. J. Brazil, "Behavioural modelling of rf power amplifiers using modified volterra series in the time domain," in *High Frequency Postgraduate Student Colloquium*, Sep. 2004, pp. 169–174.

[20] C. Crespo-Cadenas, J. Reina-Tosina, and M. J. Madero-Ayora, "Volterra behavioral model for wideband rf amplifiers," in *IEEE Transactions on Microwave Theory and Techniques*, vol. 5, no. 3, 2007, pp. 449–457.

[21] P. Wambacq, G. Gielen, P. Kinget, and W. Sansena, "High-frequency distortion analysis of analog integrated circuits," in *IEEE Trans. Circuits Syst. II: Analog Digit.Signal Process*, vol. 46, no. 3, 1999, pp. 335–345.

[22] R. Kumar, A. Banerjee, and B. Vemuri, "Trainable convolution filters and their application to face recognition," in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, 2012, pp. 1423–1436.

[23] Y. Hu and K. Muyaram, "A modified-volterra-seriestechnique for improving the accuracy of quasistatic harmonic balance analysis in coupled device and circuit simulation," in *IEEE 2004 Custom Integrated Circuits Conference*, Apr. 2004, pp. 125–128.

[24] S. Im, "Adaptive equalization of nonlinear digital satellite channels using a frequency-domain volterra filter," in *IEEE Military Communications Conference*, Apr. 1996, pp. 843–848.

[25] L. Liu, L. Li, Y. Huang, K. Cui, Q. Xiong, F. N. Hauske, C. Xie, and Y. Cai, "Intrachannel nonlinearity compensation inverse volterra series transfer function," *Journal of Lightwave Technology*, vol. 30, no. 3, pp. 310–316, 2012.

[26] G. Mitsis, M. Poulin, P. Robbins, and V. Marmarelis, "Nonlinear modeling of the dynamic effects of arterial pressure and co2 variations on cerebral blood flow in healthy humans," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 11, pp. 1932–1943, 2004.

[27] H. Ogura, "Estimation of wiener kernels of a nonlinear system and a fast algorithm using digital laguerre filters," in *15th NIBB Conference*, 1985, pp. 14–62.

[28] K. Geng and V. Z. Marmarelis, "Methodology of recurrent laguerre-volterra network for modeling nonlinear dynamic systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2196 – 2208, 2016.

[29] J. Wray and G. G. R. Green, "Calculation of the volterra kernels of nonlinear dynamic systems using an artificial neural network," *Biological Cybernetics*, vol. 71, no. 3, pp. 187–195, Jul. 1994.

[30] R. J. Campello, G. Favier, and W. C. do Amaral, "Optimal expansions of discrete-time volterra models using laguerre functions," *Automatica*, vol. 40, pp. 815–822, 2004.

[31] Y. Kang, J. Escudero, D. Shin, E. Ifeachor, and V. Marmarelis, "Principal dynamic mode analysis of eeg data for assisting the diagnosis of alzheimer disease," *Medical Imaging and Diagnostic Radiology*, vol. 3, pp. 1–10, 2015.

**Xinying Wang** Biography text here.

**Thong Nguyen** Biography text here.

**Jose E Schutt-Aine** Biography text here.