Demystifying Machine Learning for Signal and Power Integrity Problems in Packaging

Madhavan Swaminathan, *Fellow, IEEE*, Hakki Mert Torun, *Graduate Student Member, IEEE*, Huan Yu, Jose Ale Hejase, *Member, IEEE*, and Wiren Dale Becker, *Fellow, IEEE*

***Abstract*— In this article, we cover the fundamentals of neural networks and Bayesian learning with a focus on signal and power integrity problems arising in packaging. Rather than only focus on mathematical formulations, we explain the important concepts and the intuition behind them, thereby demystifying the use of machine learning for these problems. We also share some of the recent developments in this area along with future research directions in the context of packaging. Links to open-source downloadable software for some of the methods discussed are also provided.**

***Index Terms*— Bayesian learning, behavioral modeling, design optimization, neural networks, signal and power integrity (SI/PI), surrogate modeling.**

NOMENCLATURE

3-D	IC 3-D integrated circuit.
ADD-GP	Additive Gaussian process.
BAL	Bayesian active learning.
BALDO	Bayesian active learning with dropout.
BO	Bayesian optimization.
CEL	Causality enforcement layer.
CNN	Convolutional neural network.
DNN	Deep neural network.
DPTBO	Bayesian optimization with deep partitioning tree.
DSE	Design space exploration.
EI	Expected improvement.
FFNN	Feedforward neural network.
GP	Gaussian process.
I/O	Input–output.
IMGPO	Infinite metric Gaussian process optimization.
INN	Invertible neural network.
IoT	Internet of Things.
IP	Intellectual property.
IVR	Integrated voltage regulator.
LHS	Latin hypercube sampling.
LSTM	Long-short term memory.
MCMC	Markov chain Monte Carlo.
MES	Maximum entropy search.
NARX	Nonlinear autoregressive network.
NMSE	Normalized mean squared error.
NN	Neural network.
PDF	Probability density function.
PDN	Power delivery network.
PEL	Passivity enforcement layer.
PLS	Partial least squares.
PoI	Probability of improvement.
PRBS	Pseudorandom bit sequence.
PSO	Particle swarm optimization.
PTH	Plated through hole.
RNN	Recurrent neural network.
S-TCNN	Spectral transposed convolutional network.
SI	Signal integrity.
SiP	System-in-package.
SIW	Substrate integrated waveguide.
TSBO	Two-stage Bayesian optimization.
UCB	Upper confidence bound.
UQ	Uncertainty quantification.
VCO	Voltage-controlled oscillator.
WPT	Wireless power transfer.

Manuscript received June 14, 2020; accepted July 15, 2020. Date of publication July 27, 2020; date of current version August 14, 2020. This work was supported in part by the NSF under Grant No. CNS 16-24731 - Center for Advanced Electronics through Machine Learning (CAEML). Recommended for publication by Associate Editor D. G. Kam upon evaluation of reviewers' comments. *(Corresponding author: Madhavan Swaminathan.)*

Madhavan Swaminathan, Hakki Mert Torun, and Huan Yu are with the 3D Systems Packaging Research Center (PRC), School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: madhavan.swaminathan@ece.gatech.edu; htorun3@gatech.edu; huanyugt@gmail.com).

Jose Ale Hejase is with IBM Corporation, Austin, TX 78758 USA (e-mail: jhejase@ieee.org).

Wiren Dale Becker is with IBM Corporation, Poughkeepsie, NY 12601 USA (e-mail: wbecker@us.ibm.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCPMT.2020.3011910

I. INTRODUCTION

EARLY history of machine learning (ML) dates back to 1943 when the first neural networks were developed. Over the last 75 years, this field has seen bursts of research, especially after IBM's Deep Blue computer beat the world chess champion in 1997. Neural networks have been in prominence with the packaging community since the mid-2000s when these techniques were applied to the modeling of digital and microwave circuits for capturing their behavior [1], [2]. With ML gaining momentum in data science for solving problems that are otherwise unsolvable, we believe that this is an appropriate time to revisit the application of ML in packaging. Through this article, we try to provide a fundamental understanding of ML as it applies to address the SI and PI problems arising in package design. Using several examples, we try to demystify the math by stressing on

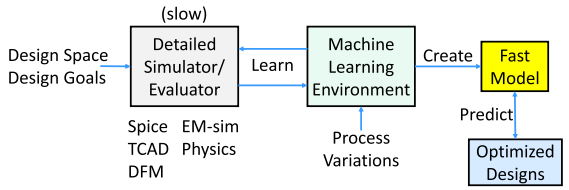


Fig. 1. Model-based design paradigm (Courtesy: E. Rosenbaum, UIUC).

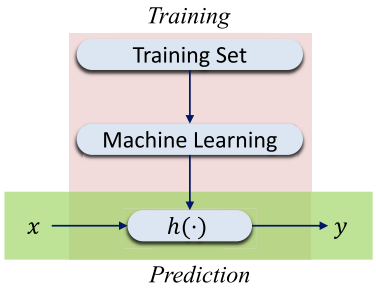


Fig. 2. Fundamentals of ML.

concepts that when logically applied can lead to the solution of problems, which can remove the "human from the loop," eliminate mistakes during the design process, reduce design respins, enable a significant reduction in design cycle time, and above all improve the overall productivity.

To start, an important question to ask ourselves is "What is ML?" The Computer Science community has been using a procedure called "data mining" for many years. In data mining, a user extracts insights from data using several techniques, such as statistical methods, data analytics, ML, and others. Therefore, ML is a subset of data mining. In data science, a technique that is often used is called statistical learning, where a user fits theoretical distributions to small sets of data based on certain assumptions. The resulting model is mathematically proven, meaning that inferences can be derived based on the model. In ML, computers are used to probe vast amounts of data for structure where the error is validated using new data. ML-based methods do not have a theoretical test as in statistical learning and therein lies the problem, meaning that just because errors have been minimized on the training data sets and validated on new data, there is no guarantee that the predictions made using the model are always accurate. However, with large amounts of data, cheap and powerful computational processing, and affordable data storage available today, the hope is that ML can be used to capture complex and nonobvious relationships that cannot be done otherwise, leading to accurate predictions.

But why is ML important for the semiconductor industry, especially in packaging? A major limit in modern electronic design automation (EDA) is design respins due to hardware complexity. Many of the failures causing respins can be attributed to insufficient modeling capability where using simulations in the design loop is oftentimes too slow and frustrating. With design complexity and performance increasing, any approximations or assumptions made during the design process can only lead to errors. In such scenarios, removing the "human from the loop" and having the machines do the work can only lead to benefits. Therefore, a model-based design paradigm can be developed where fast to evaluate "learned" model replaces the conventional "slow" model in design and design optimization. This is shown in Fig. 1 where the data from the "slow" but detailed simulator are used to develop a machine learned "fast" model. The fast model is quick to compute, is expected to have the same accuracy as the detailed simulator, and can therefore be used in the design loop for simulation-based design and optimization. Moreover, since the fast model is expected to capture the entire design space including process variations, the probability of introducing errors can be minimized. In this article, our focus is on fast model development and design optimization, two key elements that are important to developing the model-based design paradigm shown in Fig. 1.

In general, implementing ML consists of two parts, namely training and prediction, as shown in Fig. 2. Consider a 1-D function $f(\cdot)$ that maps the input x to output y. The goal is to determine the unknown function $f(\cdot)$ using which the relationship between the input and output can be determined. A set of training data $(x, y) = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is first used to capture this relationship resulting in a model $h(\cdot)$ shown in Fig. 2. It is important to note that since $h(\cdot) \neq f(\cdot)$, the derived model may be prone to error. Hence, the error between the model and training data sets need to be minimized by using appropriate amounts of data. Once complete, the learned model $h(\cdot)$ replaces $f(\cdot)$ and is used to predict the output y for a given x. This appears like a simple mapping problem, especially when the function $f(\cdot)$ is linear. But what happens if the function is highly nonlinear and is D-dimensional where $D > 5$, the output response is unknown, the input covers a large range, and vast amounts of training data are computationally expensive to obtain or unavailable, as in our SI and PI domains. In such scenarios, learning patterns from data become important, and this is where ML-based techniques become useful.

In this article, we start with an introduction to artificial NNs where the error is validated against known results. Three NNs are discussed, namely FFNN for learning the behavior of VCO; 2) RNN for replicating the behavior of input-output drivers; and 3) CNNs for capturing frequency responses. Of the NNs discussed, we believe that RNN and CNN are extremely useful for the packaging community due to their ability to address nonlinear behavior and higher dimensional problems. The outcome of the three NNs discussed results in fast models, meaning that the computational time using these models is several orders of magnitude faster than detailed models. In addition, it comes with other benefits such as protecting IP (when models need to be exchanged), compatibility with existing simulators, and so on.

We follow this with applications where the results are unknown, and hence, there is no means to minimize error. This scenario typically arises in optimization where the objective is to achieve the best performance given the design space and process capabilities. We, therefore, discuss this in the context of design cycle time reduction by introducing Bayesian

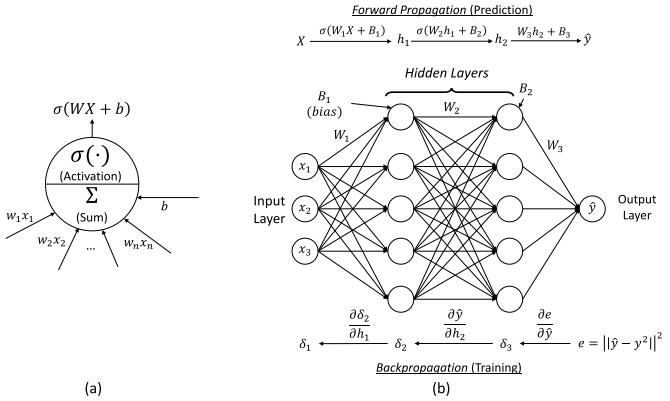


Fig. 3. (a) Single neuron. (b) FFNN.

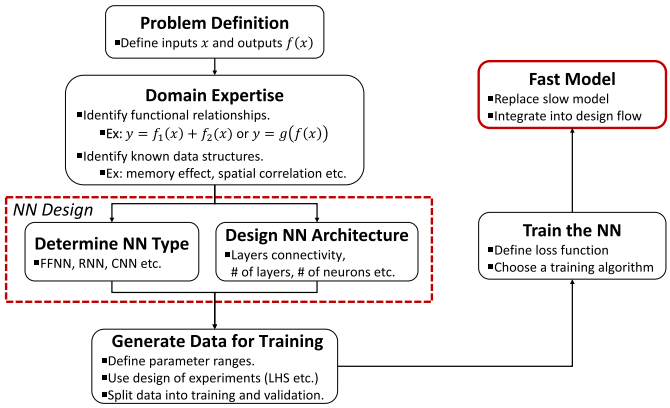


Fig. 4. Flowchart for developing a fast NN model.

learning, a method that is the most powerful and a "must have" for addressing packaging-related problems.

The question that is oftentimes asked is whether algorithms available in the public domain can be downloaded and applied readily to packaging problems. The answer is "NO" since ML is based on domain-specific assumptions, which need to be modified based on the patterns in data. We, therefore, introduce two new ML-based optimization techniques specific to packaging namely, TSBO and DPTBO, and apply it to different packaging-related problems with varying complexity.

NNs are generally overconfident models, meaning that they assume that the predictions they make are always correct. As expected, this can be dangerous. A better approach is to quantify the error in the predictions. We provide details on a learning method that can establish confidence bounds or uncertainty around predictions. We believe that such a method when combined with optimization can be very powerful, since if the uncertainty is too large, it can be reduced by adding appropriate data samples. We, therefore, introduce a technique called BALDO and apply it to high-speed channels.

For future directions, we address three important areas that are important to packaging, namely, addressing dimensionality for problems with more than 50 parameters, predicting responses outside the training range, and inverse design where the output specifications are used to derive design parameters.

The prior work in the open literature on the application of ML to SI and PI problems is limited. Through this article, we therefore provide details on the fundamental concepts and procedures behind ML for the nonexpert while providing the motivation for researchers working in packaging to further expand this field. Using the eight problems discussed in this article, we have tried to illustrate the applicability and usefulness of ML-based solutions for a variety of applications.

II. Building Fast Predictive Models Using NNs

We start with a single neuron as shown in Fig. 3(a), which represents the building block of an NN. A neuron takes a vector of inputs $\mathbf{X} = (x_1, \ldots, x_n)$, constructs their weighted sum $\mathbf{WX} = (w_1x_1, \ldots, w_nx_n)$, and adds a bias ($\mathbf{b}$) to generate $\mathbf{WX} + \mathbf{b}$, where $\mathbf{b}$ is similar to an intercept term. This is then passed through a nonlinear activation function to get $\sigma(\mathbf{WX}+\mathbf{b})$, which represents the output of this single neuron as $\sigma(\mathbf{WX} + \mathbf{b})$. The purpose of this activation function is to introduce nonlinearity and bound the output.

An NN connects several of these neurons, as shown in Fig. 3(b). A typical architecture consists of an input layer, multiple hidden layers, and an output layer. The input layer consists of the input variables (design parameters) that we want to map to an output. The purpose of the hidden layers is to capture nonobvious interactions between the overall input-output relationship. We use multiple neurons in each hidden layer and connect each one to all the other neurons in the subsequent layers, where each connection describes a different interaction pattern. As the number of hidden layers increases for capturing more complex patterns in data, the NN becomes a DNN.

The output of the output layer, i.e., NN generated output, is then compared with the actual output (training data) to calculate error (e). This error is then minimized by adjusting the weights in each layer through their gradients, and this process is known as NN training. To make the training procedure computationally efficient, the gradients are calculated through chain rule of derivatives, i.e., the gradient of weights in a particular layer is backpropagated to the previous layer, as shown in Fig. 3(a). Since the data move from input to output layer in a single direction, we call this as an FFNN. FFNNs and many backpropagation-based training algorithms are widely available as plug-and-play modules in many programming languages, such as Python and MATLAB [3], [4]. It is important to note that the number of neurons per layer and number of layers in the NN can vary based on the data sets. These can be determined through validation and optimization as briefly described in a later section.

We next discuss the modification of the FFNN to solve a simple problem based on domain knowledge, followed by two complex NNs, namely RNN and CNN, for addressing packaging-related problems. For all the problems discussed, we follow the flowchart in Fig. 4 to build a fast NN model.

A. Feed Forward Neural Networks

FFNN is the most commonly used model type in the growing field of ML for SI and PI problems. Although there is early work that uses this methodology, for instance, for high-speed interconnect [5] and embedded passive

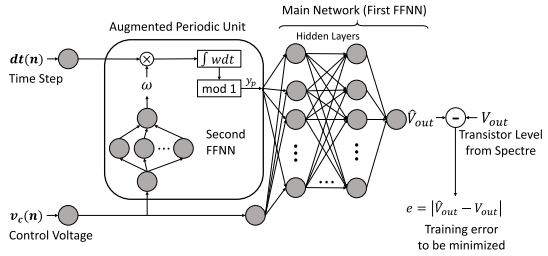


Fig. 5. Modified FFNN architecture for VCOs [12].

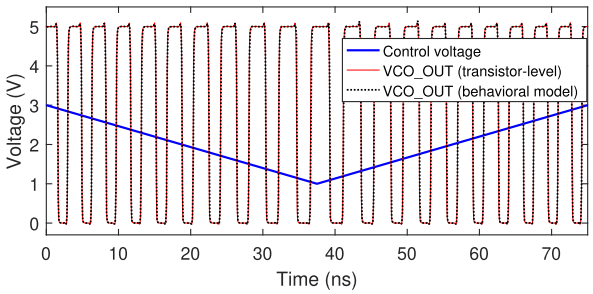


Fig. 6. Comparison of the output waveform obtained using the fast FFNN model and slow transistor model for varying control voltage [13].

analysis [6], the scope of applications that FFNNs are being used has rapidly increased in recent years. Some examples include predicting target impedance violations in PDNs [7], eye diagram modeling [8]–[10], and current prediction at package pins [11].

Here, we consider VCO to illustrate the methodology for developing and applying an FFNN. Though capturing the behavior of VCOs looks trivial, directly applying the FFNN to this circuitry will not work, and hence, modifications based on domain knowledge are required.

1) Domain Knowledge: The steady-state time-domain output waveforms of a VCO with multiple output ports can be expressed as

$$V_{\text{out}}(t) = F\left(\int \omega(t)dt\right), \quad \omega(t) = G(v_c(t)) \tag{1}$$

where $\omega(t)$ is the instantaneous frequency, V_{out} is the oscillator output voltage, $F(\cdot)$ is a periodic function that captures the shape of the waveform, $G(\cdot)$ is the function that maps control voltage $v_c(t)$ to $w(t)$, and $\int \omega(t)dt$ is the total oscillation phase. The unknown parts in this equation are two nonlinear functions, $F(\cdot)$ and $G(\cdot)$. We assume that (1) and the knowledge of workings of a VCO are available.

From (1), it can be seen that the output of a VCO ($V_{\text{out}}(t)$) is a function of the control voltage and the time step dt. First, the control voltage goes through a nonlinear function, $G(\cdot)$, to obtain $w(t)$. This is then integrated with respect to dt to obtain the oscillation phase, which goes through the second nonlinear function $F(\cdot)$ to form the output voltage waveform.

2) Network Architecture: Rather than using the generic FFNN architecture in Fig. 3, we now create an augmented FFNN architecture as shown in Fig. 5 to capture this multistage relationship [13]. Here, we use two separate FFNNs, a main FFNN and a periodic unit (PU) that contains a second FFNN. The second FFNN in Fig. 5 maps the control voltage to the oscillation frequency and represents $G(\cdot)$ in (1) as

$$\omega(n) = g_{\text{PU}}^{\text{FFNN}}(v_c(n)) \tag{2}$$

where $g_{\text{PU}}^{\text{FFNN}}(\cdot) \approx G(\cdot)$ and n is the discrete time step. The predicted $w(n)$ is then multiplied by the time steps, $dt(n)$, to calculate the phase integral as

$$y_p(n) = \left(\sum_{i=1}^{n} \omega(i)dt(i)\right) \bmod 1 \tag{3}$$

where $y_p(n)$ represents the oscillation phase and is bounded to $[0, 1)$ using the modulus operation. The output of the PU, y_p, and the control signal, v_c, are then fed into the main FFNN, which outputs the voltage waveform as

$$V_{\text{out}}(n) = f_{\text{main}}^{\text{FFNN}}(y_p(n), v_c(n)) \tag{4}$$

where $f_{\text{main}}^{\text{FFNN}}(\cdot)$ represents the main FFNN.

3) Problem 1: Consider the response of a VCO generated using a transistor-level model as in Fig. 6, where the lowest frequency is 0.216 GHz and the highest frequency is 0.306 GHz with the control voltage v_c ranging from 1 to 3 V. The response is generated using a circuit simulator, Spectre [14], which is used as the training data [13].

The training process uses the aforementioned backpropagation method to adjust the weights of all neurons until the error between the transistor-level circuit model and the NN model is minimized, as shown in Fig. 5. Using 20 neurons in the first, 10 neurons in the second hidden layer of the main FFNN, and 5 neurons in the hidden layer of the second FFNN, the entire behavior of the VCO can be captured over the voltage range of 1–3 V, as shown in Fig. 6.

Once the NN model is created, it can be incorporated into commercial simulators in the form of an input-output module, Verilog-A [12]. This module can now be used as a behavioral model to replace the transistor-level circuit and for protecting IP since such models cannot be reverse-engineered. An additional benefit of creating a behavioral model is that they simulate much faster than a transistor-level model. For the VCO example, the fast NN model provides a 93% reduction in simulation time [13]. A question that is often asked is: how does one determine the number of neurons and hidden layers required. The only way to determine this in this example, and others, is through experimentation.

B. Recurrent Neural Networks

FFNNs, though powerful, are not very useful in several time-domain problems we encounter in SI for packaging. This is because in most of the SI problems, the output of a circuit at the present time depends on the outputs at the previous time. As FFNNs operate in a single direction, they cannot capture this behavior. To overcome this problem, we feed the NN predicted output back to its input as in Fig. 7 to predict the next output. The NN architectures with this kind of a feedback loop are called RNNs and are very useful for predicting the output behavior of input-output circuits. As such, their application

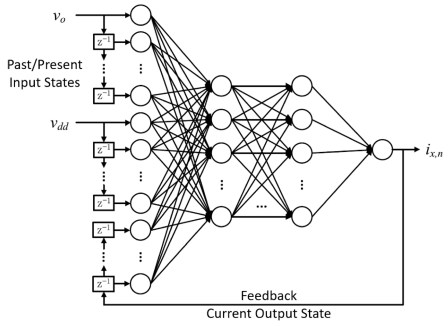


Fig. 7. NARX RNN architecture for modeling input-output drivers [20].

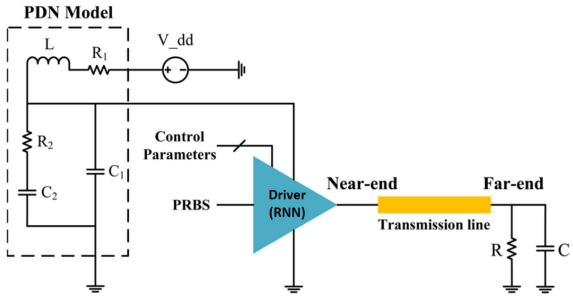


Fig. 8. Input-output driver with package parasitics [20].

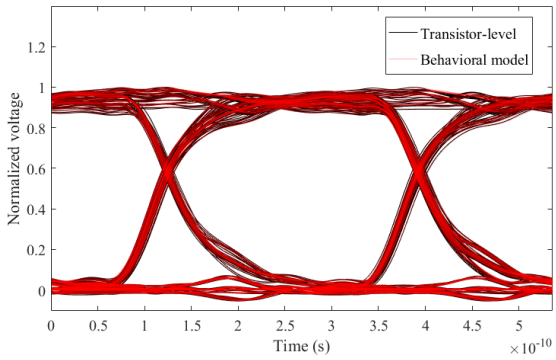


Fig. 9. Eye diagram at the far end of the transmission line in Fig. 8, obtained with RNN model and transistor-level simulations [20].

to modeling of transient nonlinear behavior of input-output buffers have been previously explored in [15]–[18].

1) Domain Knowledge: We use the example of input-output drivers to illustrate the application of RNNs. Behavioral models of input-output drivers can be created by modeling the relationship between current and voltage at the output port and the power supply port [19]. The driver output current can be written as

$$i_o(t) = w_{o,H}(t)\ i_{o,H}(t) + w_{o,L}(t) i_{o,L}(t) \tag{5}$$

where $i_{o,H}(t)$ and $i_{o,L}(t)$ are called submodels and represent the output current of the driver for high and low logic states, respectively, and $w_{o,H}(t)$ and $w_{o,L}(t)$ are the corresponding weighting functions. Similarly, the current at the power supply port can be written as

$$i_{\rm dd}(t) = w_{{\rm dd},H}(t)\ i_{{\rm dd},H}(t) + w_{{\rm dd},L}(t)\ i_{{\rm dd},L}(t) + \delta_{\rm dd}(t) \tag{6}$$

where $i_{{\rm dd},H}(t)$ and $i_{{\rm dd},L}(t)$ are the submodels of the supply port for driver input high and low, and $\delta_{\rm dd}(t)$ represents the crowbar supply current component during logic switching. In (5) and (6), the weighting functions scale the contributions from the corresponding submodels and can be extracted as explained in [20].

These current submodels can be compactly defined as

$$i_{v,n}(t) = i_{v,n}(v_o(t), v_{\rm dd}(t), \mathbf{D}) \tag{7}$$

where $v = \{o, {\rm dd}\}$ and $n = \{H, L\}$ denote the port and logic state configurations, respectively, $v_o(t)$ is the output voltage, $v_{\rm dd}(t)$ is the supply voltage, and $\mathbf{D}$ represents the influence of the previous voltage and current values, i.e., the memory effect. To create the behavioral model of input-output drivers, we create RNN models to replace the submodels $i_{v,n}(t)$ to capture the nonlinear dynamic current–voltage relationship. As in VCOs, we assume that (5)–(7) are known and the data defining these equations are available through a slow transistor-level model.

2) Network Architecture: A particularly useful type of RNN that can capture the memory effect is NARX, as shown in Fig. 7 [20]. Here, the output of the RNN depends on the current input state, past input states, and past output states.

The inputs to the NARX RNN model for input-output drivers include voltage and current at previous time steps and its own output at previous time steps (through feedback). The submodels in (7) can therefore be written using the RNN as

$$i_{v,n}(t) = i_{v,n}^{\rm RNN}\begin{pmatrix} v_o(t), v_o(t-h), \ldots, v_o(t-rh) \\ v_{\rm dd}(t), v_{\rm dd}(t-h), \ldots, v_{\rm dd}(t-rh) \\ i_{v,n}(t-h), \ldots, i_{v,n}(t-rh) \end{pmatrix} \tag{8}$$

where h is the sampling time step and r is referred to as the dynamic order of the model, which usually has a value of 1 or 2 for typical drivers.

To generate the data to train the RNN architecture, we use the transistor-level models of the input-output drivers, where we connect voltage sources at the driver output [both $v_{\rm dd}(t)$ and $v_o(t)$]. We then simulate the output current when the input to the driver is held at logic 1 or 0 to cover a wide range of variability as explained in detail in [20]. The switching waveforms are also used to learn the weighting functions.

3) Problem 2: Let us now apply the RNN model for an industrial-strength driver circuit with preemphasis. The simulation setup is shown in Fig. 8, where a driver is connected to a 50-Ω transmission line with a length of 50 mm and terminated with a 60-Ω resistor and a 0.7-pF capacitor. A PDN model in the form of R, L, and C is used to provide nonideal power supply. The training data are generated using the PDN-aware transistor-level driver model where the supply and output ports are excited using a total of 67k training waveforms for various combinations of input bit sequences, with further details in [20]. The resulting output waveforms are then used along with current/voltage relationship to train the RNN, which then is implemented as a Verilog-A model [20].

As can be seen in Fig. 9, the eye diagrams generated from the RNN fast model agree well with the transistor model.

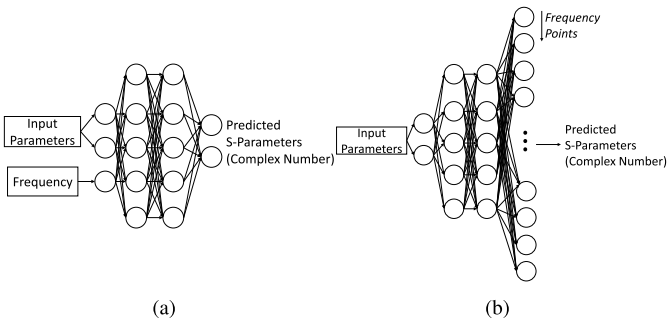


Fig. 10. FFNN using (a) frequency as input and (b) frequency as output.

The resulting speedup in computing the eye diagram (500-bit long PRBS with 268-ps bit period) using the RNN model is 300×. The RNN model can now be used to replace the original circuit model to protect IP and for fast simulations when transmission line parameters, such as impedance and parasitics, change in Fig. 8.

C. Convolutional Neural Networks (Transposed)

In SI and PI problems, computing the frequency response is very important. Therefore, given a structure with a set of variable parameters, an important question to address is whether parameterized frequency responses can be learned using NNs. Two possible FFNN architectures are shown in Fig. 10.

Let us start with simple calculations. Consider the response of a four-port structure at 2000 frequency points, parameterized with respect to ten variables. To train an NN, we start by creating data, where we compute, say, 500 variable combinations and store their corresponding responses. For the frequency as input configuration in Fig. 10(a), we need to replicate each variable combination 2000 times to generate an input-output data set, resulting in 1M data points. This creates a significant memory problem for training algorithms due to the large data size.

Another possibility is to make each frequency point a separate output dimension as in Fig. 10(b). For an N-port reciprocal structure evaluated at M frequency points, the total number of dimensions (outputs) is $D = 2MN(N+1)/2$, where the factor of 2 comes from real and imaginary parts of a complex number. Consider a very simple FFNN that has only one hidden layer with ten neurons. The number of weights that connect this hidden layer to D outputs is then $10D$. For a four-port structure at 2000 frequency points, D becomes 40k and the number of weights for just a single hidden layer is 400k. Although the memory problem is reduced, learning that many weights can be challenging and can lead to overfitting. Overfitting is caused when random fluctuations in the training data are learned as being part of the model, which leads to erroneous predictions. A familiar example to overfitting is using a very high-order polynomial when the data can be better described by a lower order polynomial.

As both approaches do not scale to practical SI and PI problems, we use our domain knowledge on frequency responses to replace NNs in Fig. 10 with a CNN. Currently, there are only a few studies that use CNNs in SI and PI domain. Recent work includes using CNNs for decoupling capacitor optimization [21] and electromagnetic interference prediction [22]. To the best of our knowledge, CNNs have not been explored in the SI and PI domain to handle frequency responses. In the following, we discuss the development of CNN-based models to address the drawbacks of FFNNs when handling frequency responses.

1) CNN: Let us consider Fig. 10(b). Although the frequency at the output becomes a high-dimensional problem, it is structured, meaning that there exists a spatial correlation along the frequency axis, where neighboring frequency points are highly correlated with each other. This spatial correlation can be exploited using a CNN. The hidden layers of a CNN are called convolutional layers, and unlike the NNs considered so far, the neurons in each layer do not connect to all other neurons in the subsequent layers. The goal here is to learn local patterns in the axes that contain spatial correlations.

In the context of frequency responses, this corresponds to searching for patterns, such as resonances and ripples in smaller frequency bands. CNNs achieve this by stacking multiple convolutional layers. Unlike fully connected layers where each neuron has a single weight, the neurons in convolutional layers form 1-D arrays (also called kernels).

Let $\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$ be the m-dimensional input vector and $\mathbf{h} = [w_1, w_2, \ldots, w_k]^T$ be the convolution kernel of size k. The downsampling operation done by a single kernel can be written as

$$y = f(\mathbf{h} * \mathbf{x}) = f(\mathbf{H}\mathbf{x}) \tag{9}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \ \mathbf{H} = \begin{bmatrix} w_1 & w_2 & \cdots & w_k & 0 & \cdots & 0 \\ 0 & w_1 & w_2 & \cdots & w_k & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & w_1 & w_2 & \cdots & w_k \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \tag{10}$$

where $(*)$ denotes the convolution operation, $f(\cdot)$ is the nonlinear activation function, $\mathbf{H}$ is the convolution matrix of size $n \times m$, and $\mathbf{y}$ is the downsampled output of size $n = m - k + 1$. When the convolution operation is written as $f(\mathbf{H}\mathbf{x})$, each row of H represents the frequency axis. As the learnable weights $(w_{1,\ldots,k})$ in each row are the slid version of the same values, the weights are shared across the frequency axis.

2) Transposed CNN: In a CNN, convolutional layers are used to downsample high-dimensional inputs to low-dimensional features similar to the visual cortex in the brain. To predict frequency responses, we need upsampling operations as low-dimensional design parameters are at the input side and the high-dimensional frequency response is at the output side. To achieve this, we make use of transposed convolutional layers. These are learnable upsampling layers that preserve the spatial correlation in its output. More formally, for an n dimensional input vector $\mathbf{x}$, transposed

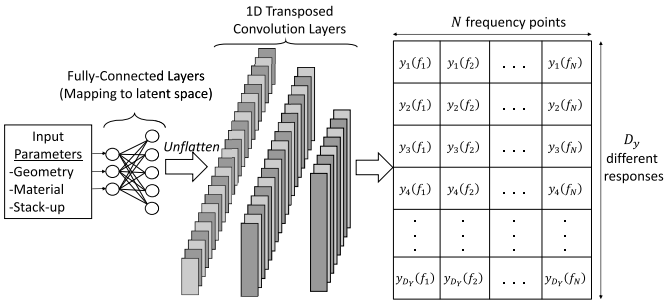


Fig. 11. S-TCNN.

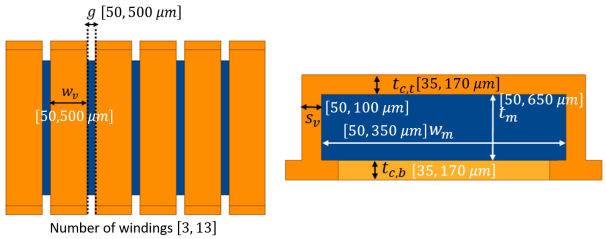


Fig. 12. Solenoid inductor [24]. (a) Top view. (b) Side view.

convolution operation can be written as

$$\mathbf{y} = f(\mathbf{h} *^{\mathsf{T}} \mathbf{x}) = f(\mathbf{H}^{\mathsf{T}}\mathbf{x}) \tag{11}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{H}^{\mathsf{T}} = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ w_2 & w_1 & \ddots & \vdots \\ \vdots & w_2 & \ddots & 0 \\ w_k & \vdots & \ddots & w_1 \\ 0 & w_k & \ddots & w_2 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & w_k \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{12}$$

where $*^{\mathsf{T}}$ is the transposed convolution operation and $\mathbf{y}$ is the upsampled output of size $m = n+k-1$. Note that upsampling ratio can be increased by making use of strided transposed convolutions, which is not covered here, but described in [23].

One question that can be asked here is: how do we know if an arbitrary design parameter can be upsampled to obtain the corresponding frequency responses? The answer is we do not and, therefore, we first convert the inputs to a latent space that best describes the start of the sequence using fully connected layers in the first few layers in our network. We call the resulting architecture as S-TCNN [24] and is shown in Fig. 11. An open-source Python implementation of S-TCNN can be found at `https://github.com/GT-PRC/S-TCNN`.

To train the S-TCNN model, we use the conventional backpropagation method to minimize the error

$$L_{\text{freq}} = \frac{1}{N}\sum_{n=1}^{N}\sqrt{\frac{1}{K}\sum_{k=1}^{K}(y_{n,k} - \hat{y}_{n,k})^2} \tag{13}$$

where $\hat{y}_{n,k}$ is the predicted output at the kth frequency point for the nth training sample. It is very important to note here that the error in (13) is significantly different than conventionally used mean-squared error (MSE) as explained in detail in [24]. The error in (13) ensures that the S-TCNN model learns a mapping from the input vector to the frequency response as a whole, rather than the mapping to k different frequency points.

3) Problem 3: Consider solenoidal inductors used in IVR modules [24]. The geometry of the inductor is shown in Fig. 12, which consists of copper windings around a magnetic core. The eight physical parameters and their ranges ([Min, Max]) are shown in Fig. 12. The goal is to map the eight input parameters to inductance and resistance of the inductor over the frequency range 10–500 MHz.

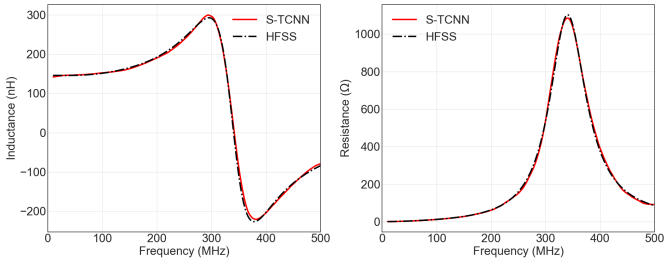


Fig. 13. Comparison of S-TCNN predicted (a) $L(f)$ and (b) $R(f)$ to HFSS [24].

To create the training data, we generate 800 samples based on LHS and feed into a 3-D electromagnetic (EM) solver, Ansys HFSS, to extract the frequency-dependent inductance and resistance at 200 frequency points between 10 and 500 MHz. LHS is a well-known and structured sampling (design of experiments) scheme that fixes the values of the input parameters for computing the output response that covers the entire sample space [25].

As shown in Fig. 13, the trained S-TCNN model can accurately capture the resonant inductor behavior. The predictive error of the model, when calculated using a modified NMSE metric for 200 validation samples as in [26], is found to be 12.0%. Validation samples constitute the data samples not used for training but instead used to check the accuracy of model predictions. These results are compared with the known outputs to validate the accuracy of the ML model.

In this example, it took approximately 19.2 CPU hours to collect the training data and 2 min to train the S-TCNN model. Once trained, the S-TCNN model can generate 1000 different frequency responses in ∼1.1 s compared with ∼24 h using the HFSS model.

D. Physically Consistent S-TCNN

Oftentimes, we need to constrain the NN predictions to ensure that they do not violate any physical phenomena. One common scenario is when predicting the S-parameters of package interconnects, we need to satisfy passivity and causality. To achieve this, we add two additional layers to the S-TCNN, namely the CEL and PEL, resulting in the architecture shown in Fig. 14 [26].

The CEL uses the Hilbert transform to relate the real and imaginary parts of the S-parameters, whereas the

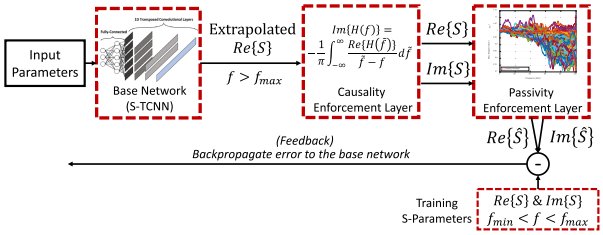


Fig. 14. Network architecture to ensure physical consistency.

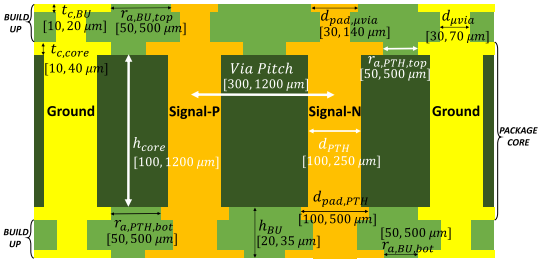


Fig. 15. Parameters of the differential PTH in package core [26].

PEL ensures that the singular values of the S-parameters are less than 1 [26], both derived from the domain knowledge on behavior of passive structures. The key to this NN architecture is the feedback loop shown in Fig. 14, where the weights are automatically adjusted as part of the learning process to ensure that the constraints are satisfied while simultaneously minimizing the error in the response.

1) Problem 4: We consider modeling a differential PTH pair in package core along with the microvias that connect to build-up layers. Here, the goal is to map 13 input parameters in Fig. 15 to their corresponding four-port S-parameters from dc to 100 GHz with 100-MHz steps, corresponding to an output dimensionality of 12 000. We determine 550 samples using LHS and extract their S-parameters using Ansys HFSS [27], which are then used for training an S-TCNN model and an S-TCNN + CEL + PEL model. We use 130 independent validation samples to evaluate the accuracy of the models.

The results show that both models correlate well with HFSS as shown in Fig. 16, where the NMSE on validation samples is calculated as 5.32% for S-TCNN and 5.47% for S-TCNN + CEL + PEL. We then calculate the maximum singular values of the predicted S-parameters for each validation sample. Fig. 17(a) shows that S-TCNN predicted S-parameters oftentimes have passivity violations that result in nonphysical predictions. On the other hand, Fig. 17(b) shows that S-TCNN + CEL + PEL completely eliminates these violations and guarantees the predicted S-parameters to be physically consistent. In terms of run times, the trained S-TCNN + CEL + PEL takes ∼0.53 s to generate 1000 different broadband S-parameters compared with ∼60 h with HFSS.

III. Design Cycle Time Reduction Using Optimization

In Section II, we first started by understanding the behavior (domain expertise), then designed a neural network to capture the behavior (architecture), and predicted the output response

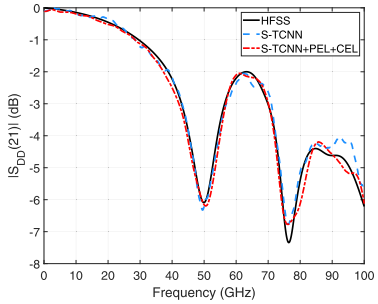
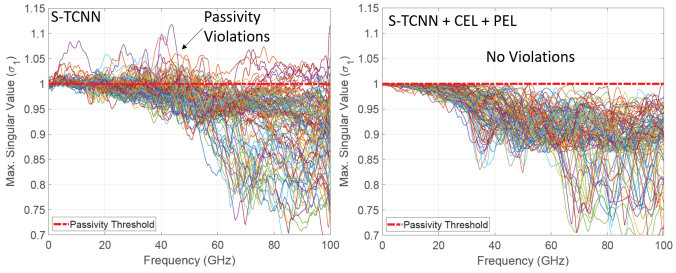


Fig. 16. Predicted differential insertion loss compared with HFSS [26].



(a) (b)

Fig. 17. Passivity of predicted S-parameters (a) with and (b) without CEL and PEL [28].

for a set of inputs. In all of the examples considered, we first generated training data and then minimized the error between the NN output and actual output. We then validated the results by comparing the predicted response with the actual response on a different data set.

At this point, it is important to ask the following questions.

1) What happens if we do not have a training data set, or in other words, we start with zero training data?
2) How many training samples do we then need to develop an accurate model?
3) Since the actual data to compare against is unavailable, how do we know that the predictions are accurate?
4) Is there a way to establish a level of confidence in the solution?

We try and answer these questions in this section in the context of design optimization for SI and PI, followed by the development of models with confidence bounds in Section IV. The predictions from ML models being considered here are not assumed to be always correct as compared to NNs described in Section II. Instead, they are uncertainty quantified models that provide an error bar in the predictions.

Consider a typical design cycle used for ensuring signal quality in a package, as shown in Fig. 18. We start with some design objectives and performance metrics and follow that with DSE leading to an initial design, which then is converted to a layout on which we do validation. During the design cycle, we iterate multiple times (shown as loops in Fig. 18) be it with DSE, initial design, or parameter tuning, which takes considerable time (both due to computations and human intervention). The "human in the loop" is important for making design decisions and ensuring accuracy but results in

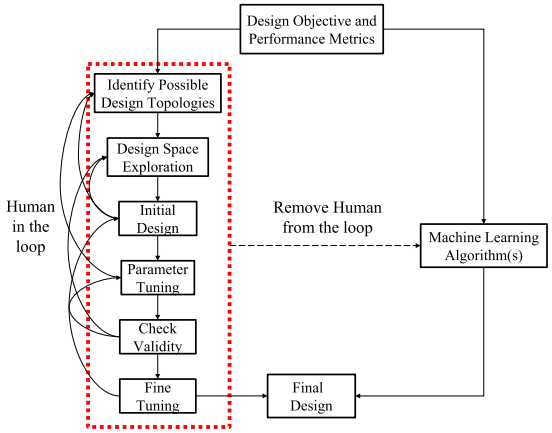


Fig. 18. Design cycle time reduction using ML.

increased computations and delayed design closure. In addition, humans are prone to making mistakes, and as a result, design respins cause increased delays. Instead, we believe that ML can remove the human from the loop as shown in Fig. 18, make intelligent design decisions, generate optimized designs, and eliminate errors, leading to significantly reduced design closure time. We address ML models in the context of design optimization for design cycle time reduction in this section.

A. Active Learning for Optimization

Consider a 1-D problem where x is the input and $f(x)$ is the output with $f(\cdot)$ being a function, where the goal is to determine the value of x that minimizes (or maximizes) the function $f(x)$, written as

$$\widetilde{x} = \underset{x \in \mathbb{X}}{\arg\min}\ f(x) \tag{14}$$

where $\widetilde{x}$ is the best input parameter in the design space, $\mathbb{X}$. If $f(x)$ is convex with a clear global optima, most gradient-based optimizers will find $\widetilde{x}$ very quickly.

However, if $f(x)$ is nonconvex and has many local minima, most optimizers will fail since they will get stuck in local minima. The nonconvex optimization problem becomes very difficult when we are unable to construct a response surface, and the design space is large and is high-dimensional. Most SI and PI problems in packaging fall under this category, and hence, there is a need for ML-based techniques.

Let us start the discussion with a set of six samples collected from a 1-D function $f(x)$. We can then fit a curve to these samples as shown in Fig. 19(a) and locate the global minimum of the curve shown in the histogram in Fig. 19(a). Then, multiple curves can be used to connect the same six samples together, each one having a different location at which the minimum occurs, as shown in Fig. 19(b). When we increase the number of such curves to infinity, a distribution of curves results along with a continuous distribution for the histogram as in Fig. 19(c). We call the histogram the probability distribution over the location of the minimum, $P_{\min}(x)$.

To place this example in the context of SI and PI analysis, assume that the six samples are generated from a computationally expensive full-wave 3-D EM solver. Let us refer to the mean of infinitely many curves as the surrogate model (a substitute to the actual model), their variation between the samples as the uncertainty of the surrogate and $P_{\min}(x)$ as the acquisition function. Instead of optimizing $f(x)$, an easier problem could be to optimize the acquisition function to find the most likely location of the global minima $\widetilde{x}$ and then perform an EM simulation at that point to observe $f(\widetilde{x})$. We can now construct a new surrogate model using seven samples and reduce the uncertainty at which minimum occurs. We can repeat this process and obtain more observations of $f(x)$ and further reduce the uncertainty, finally converging to the true location of the global minimum while using a minimum number of EM simulations as each sample is selected intelligently. Hence, a better way to solve (14) is by rewriting it as

$$\mathbf{x}_{t+1} = \underset{\mathbf{x} \in \mathbb{X}^D}{\arg\max}\ u(\mathbf{x}) \tag{15}$$

where $u(\cdot)$ is the fast-to-evaluate acquisition function and $\mathbf{x}_{t+1}$ is the point that maximizes $u(\cdot)$ in the D-dimensional input space, $\mathbb{X}^D$. This formulation recasts the difficult optimization problem in (14) as a series of easier problems that could be solved iteratively until we converge to $\mathbf{x}_{t+1} \approx \widetilde{\mathbf{x}}$.

We call this process active learning for optimization as shown in Fig. 20, where, at each iteration, a new sample is added (from the 3-D EM solver) based on the maximum of $u(\mathbf{x})$ as in (15). In Fig. 20, the role of the model is to create a distribution of functions that enable calculation of statistical metrics. The acquisition function uses these statistical metrics and acts as the "decision making" arm to determine where to query the function. Each time a new sample is collected, the surrogate model [thereby $u(\mathbf{x})$ in (15)] is updated to reflect the newly obtained information. As such, at the tth iteration, the model uses all the available data, $(\mathbf{X}_t, f(\mathbf{X}_t)) = \{(\mathbf{x}_1, f(\mathbf{x}_1)), \ldots, (\mathbf{x}_t, f(\mathbf{x}_t))\}$, to select the next sampling location, $\mathbf{x}_{t+1}$. The function is then queried at $f(\mathbf{x}_{t+1})$ to collect the new information and the process is repeated in an iterative manner.

Since the model we need has to capture a distribution of curves, we make use of a Bayesian (probabilistic) model and refer to the active learning scheme in Fig. 20 as BO. In particular, we define each curve that fits our samples in Fig. 19(b) to be a sample from a GP. A GP is a probability distribution over functions (and not data). Unlike classical distributions where a sample corresponds to a single number, a sample from a GP is a function and can be written as

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{X}_t), K(\mathbf{X}_t, \mathbf{X}_t)) \tag{16}$$

where $\mu(\mathbf{X}_t)$ is the mean function and $K(\mathbf{X}_t, \mathbf{X}_t)$ is the covariance matrix that captures relationships between variables (evaluated at training inputs). For the purposes of BO, the mean function is generally set to a constant around the empirical mean of the targets, $\mathbf{Y}_t$ [31]. Each element of the $t \times t$ covariance matrix is constructed using a kernel function, $k(\mathbf{x}_i, \mathbf{x}_j)$. Among many kernel choices [32], a useful kernel

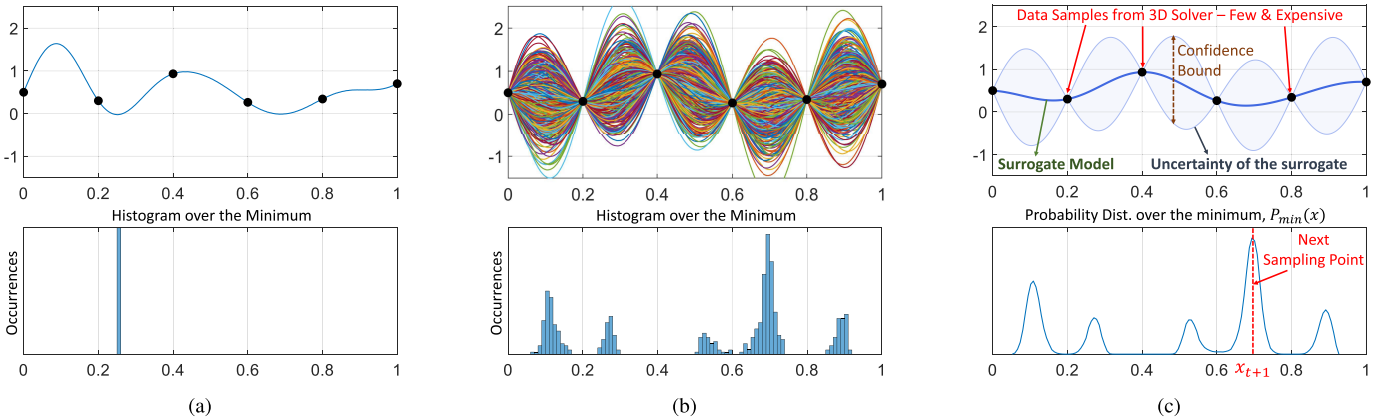


Fig. 19. Function samples and histogram over minimum. (a) Single function sample and minimum. (b) Many function samples and minima. (c) Distribution of infinitely many functions samples and minima showing surrogate model and acquisition function. Modified from [29].

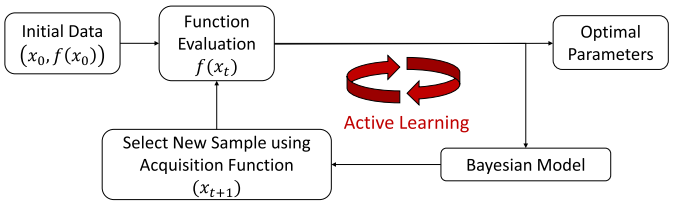


Fig. 20. High-level summary of BO (modified from [30]).

for SI and PI problems is Matern 5/2 function, given as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) e^{-\sqrt{5}r}$$
$$r = \left(\sum_{d=1}^{D} \frac{(x_{i,d} - x_{j,d})^2}{\lambda_d^2}\right)^{1/2} \tag{17}$$

where $x_{\cdot,d}$ is the dth dimension of input parameter vector, λ_d is called the length scale of each input parameter, and σ_f is a scaling factor. The parameters of this kernel function, λ_d and σ_f, are collectively referred to as the hyperparameters of the GP model, θ. Similar to finding weights of an NN as described in Section II, the training of the GP is done to determine θ through minimizing a loss function as explained in [32].

We can now use the trained GP to construct the surrogate model with confidence intervals as in Fig. 19(c). We will use the mean of the trained GP as our surrogate model, which corresponds to mean of infinitely many function samples, and the variance of it to represent the uncertainty in our predictions (envelope of function samples). The mean and variance of the trained GP can be written as [32]

$$\widehat{\mu}_\theta(\mathbf{x}^*) = K(\mathbf{x}^*, \mathbf{X}_t) K^{-1}(\mathbf{X}_t, \mathbf{X}_t) \mathbf{Y}_t \tag{18}$$
$$\widehat{\sigma}_\theta^2(\mathbf{x}^*) = k^* - K(\mathbf{x}^*, \mathbf{X}_t) K^{-1}(\mathbf{X}_t, \mathbf{X}_t) K(\mathbf{X}_t, \mathbf{x}^*) \tag{19}$$

where $\mathbf{x}^*$ is the test point, $k^* = k(\mathbf{x}^*, \mathbf{x}^*)$ is as in (17), and θ subscript indicates the dependence on hyperparameters that are obtained through training the GP.

We now need a sampling strategy to lead the BO framework to find the global maximum. A logical choice is to select the next sampling point to be the maximum of the surrogate model, $\mu(\mathbf{x})$. However, as we also want to minimize the uncertainty around the surrogate model, we sample the point with maximum uncertainty, $\sigma(\mathbf{x})$. A popular acquisition function that balances these is called UCB [33], given as

$$u_{\text{UCB}}(\mathbf{x}) = \mu(\mathbf{x}) + K\sigma(\mathbf{x}) \quad K = \sqrt{2ln(2\pi t^2/(12\eta))} \tag{20}$$

where K is the balancing constant, $(1-\eta)$ is the probability of converging to the global optima, t is the number of iterations, and $\mu(\cdot)$ and $\sigma(\cdot)$ is as in (18) and (19). The argmax of (20) is then the best balancing point that can be used as $\mathbf{x}_{t+1}$. This optimization of the $u_{\text{UCB}}(\mathbf{x})$ to obtain $\mathbf{x}_{t+1}$ is called auxiliary optimization. Here, any optimizer can be used as the gradient of the acquisition function can be calculated in closed form from (20) and it is very fast to evaluate. Other strategies that can also be used in SI and PI problems are EI and PoI [34].

The flow of BO with UCB for maximizing a 1-D nonconvex unknown response is given in Fig. 21. It can be seen that the sampling is focused on the promising regions of the function to find its maximum rather than spread over the entire sample space as would be done for nonactive learning methods.

BO is a widely used method in a variety of domains such as neuroengineering, aerospace engineering, and material science and has recently received attention in the SI and PI domain [35]–[38]. Though powerful, BO methods used in other domains need modifications before they can be applied to SI and PI problems. A few questions to ask ourselves before we continue are the following: 1) how does one select the right acquisition function? 2) how to handle a large design parameter space? and 3) how does one scale to high-dimensional problems since surrogate models in general do not scale well with increasing dimensionality.

We now present two new BO methods developed to answer these questions for optimization in the context of SI and PI, namely, TSBO [39] for problems with moderate dimensionality ($D \approx 10$) and DPTBO [31] for problems with high dimensionality ($D > 10$).

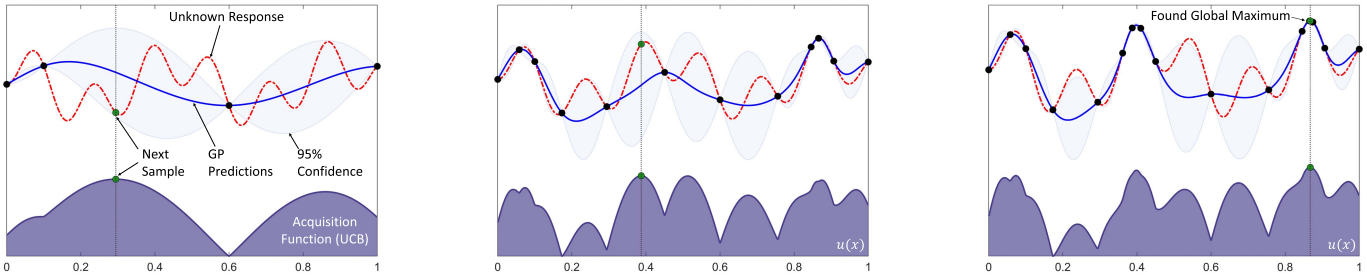


Fig. 21. BO flow for maximizing a 1-D function using the UCB acquisition function. (a) $t = 5$. (b) $t = 13$. (c) $t = 20$.

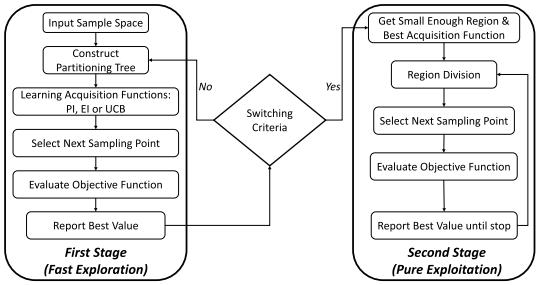


Fig. 22. Flowchart of the TSBO algorithm [39].

B. Two-Stage Bayesian Optimization

We start by separating the optimization process into coarse and fine-tuning schemes typically used by a designer to address large sample spaces. We, therefore, break-up BO into two parts as shown in Fig. 22, leading to TSBO.

The purpose of the first stage of TSBO, fast exploration stage, is to rapidly find a region in the large sample space that contains the global optimum $\tilde{\mathbf{x}}$ in (14). This is achieved by employing a novel hierarchical partitioning tree. Here, the original large sample space $\mathbb{X}^D$ is divided into 2^D subregions, as shown in Fig. 23. Center points of each subregion (referred to as candidate points) are then evaluated using $u(\mathbf{x})$, instead of the actual function $f(\mathbf{x})$. The candidate point that maximizes $u(\mathbf{x})$ is then selected as the next sampling point, $\mathbf{x}_{t+1}$, and only this is evaluated using $f(\mathbf{x})$. The subregion that $\mathbf{x}_{t+1}$ belongs to is then further divided, increasing the number of candidate points to $t2^D$ at the tth iteration. Such partitioning tree enables generating many candidate points to cover the large sample space without any additional queries to $f(\mathbf{x})$ and, hence, reduces CPU time to converge by reducing the number of EM simulations required.

A major dilemma that often arises in BO is determining which acquisition function to use since each has advantages [31], [40] and there is no way of knowing *a priori* which will perform better for a given problem. In TSBO, we learn which acquisition function is the best (from a library of strategies) for each optimization problem. This is done by observing how each acquisition function behaves in the initial stages of optimization and then selecting the one that performs the best. This extends applicability of TSBO to different design problems arising in SI and PI since it adapts its behavior based on the problem. For brevity, we refer readers to [39] for a detailed description.

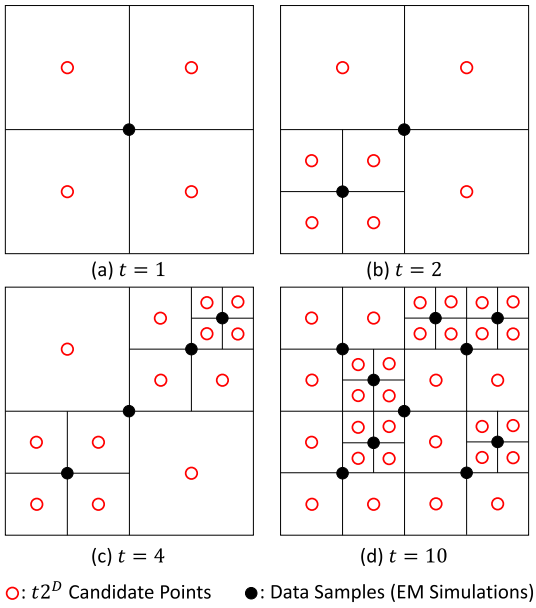


Fig. 23. Partitioning tree strategy to cover large sample spaces. (a) $t = 1$. (b) $t = 2$. (c) $t = 4$. (d) $t = 10$.

The exploration stage continues until a small enough region that contains the global optima of $f(\mathbf{x})$ is identified as explained in [39]. The second stage (pure exploitation) uses this small enough region and the learned $u(\mathbf{x})$ from the first stage, and then performs fine-tuning to increase optimization accuracy.

As an example, we apply TSBO to maximize the 2-D peaks function that is available in MATLAB. As shown in Fig. 24(a), the optimization starts with only one data point. As the algorithm progresses into 20 iterations, candidate points start to cover the entire sample space, but the active learning strategy directs function evaluations to concentrate at interesting regions, i.e., near local and global maxima. The first stage automatically ends after 42 iterations and the small region [red rectangle in Fig. 24(c)] is passed to the second stage to perform fine-tuning until a predetermined number of simulations is completed. An open-source MATLAB implementation of TSBO can be found at `https://github.com/GT-PRC/TSBO`.

We now provide two examples to show how TSBO performs against a state-of-the-art BO solution developed in other communities for practical SI- and PI-related problems.

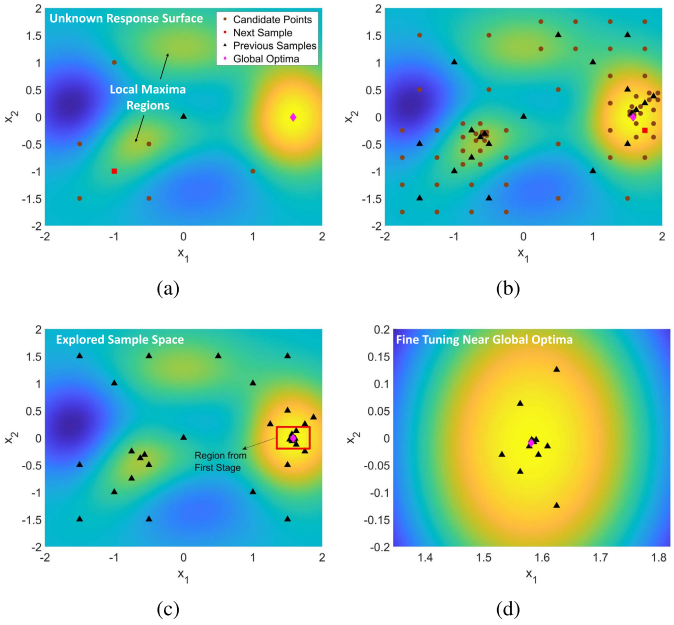


Fig. 24. Progression of TSBO for optimizing the 2-D peaks function [39]. (a) Starting point, $t = 1$. (b) $t = 20$. (c) End of first stage, $t = 42$. (d) End of second stage, $t = 100$.

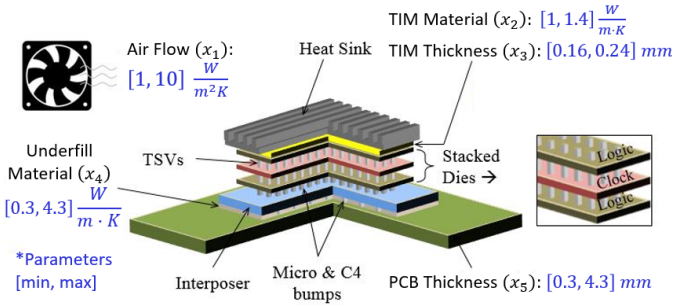


Fig. 25. Geometry of the stacked 3-D IC [42].

1) Problem 5—Clock Skew Minimization for 3-D ICs: Consider three stacked dies as in Fig. 25 where a clock distribution in the center die is sandwiched between two logic dies. As the logic circuits switch and generate heat, a temperature gradient across the center die causes clock skew. The objective is therefore to minimize clock skew by tuning five control knobs shown in Fig. 25, namely airflow velocity (x_1), thermal interface material (x_2), its thickness (x_3), underfill material (x_4), and PCB thickness (x_5). We pose this as a black-box problem, meaning that the function $f(\cdot)$ is unknown and all we can do is to query $f(\cdot)$ using the five input parameters ($x_{1,\ldots,5}$) and observe the clock skew. The function query here corresponds to a multiphysics simulation where the data $f(\cdot)$ are generated using an electrical–thermal solver [41].

The results of the optimization are shown in Fig. 26. We compare TSBO against a state-of-the-art BO algorithm, IMGPO [42], and a non-ML optimizer, namely nonlinear solver. Optimization using TSBO resulted in a clock skew of 86.0 ps compared with 88.0 ps with IMGPO and 96.6 ps with nonlinear solver. In this example, TSBO shows a much better convergence rate than either nonlinear solver or IMGPO

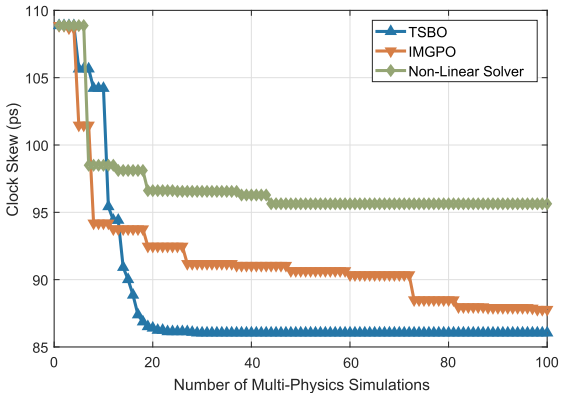


Fig. 26. Convergence of TSBO for clock skew minimization [39].

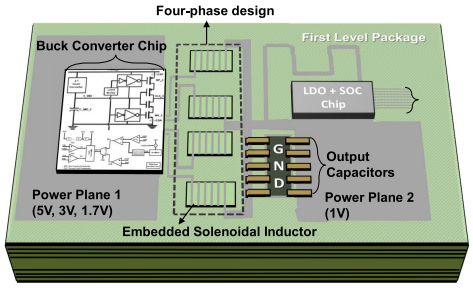


Fig. 27. Four-phase SiP IVR [39].

with a speed improvement of 3.76× and 3.96×, respectively, while providing a design with minimum clock skew.

2) Problem 6—IVR Optimization: In this example, we consider the chip-package codesign of an SiP-based IVR module. We use the solenoidal inductor from Fig. 12 to design a four-phase IVR as shown in Fig. 27, with further details available in [43]. Here, we pose the codesign problem as a black-box optimization, where the goal is to find the ten geometrical parameters of the inductor (horizontal and vertical loading ratios of magnetic material in addition to the eight parameters in Fig. 12 [39]) to maximize IVR efficiency while minimizing the inductor area. The objective function is therefore formulated as maximizing the weighted sum of efficiency and inductor area, given as

$$f(\mathbf{y}) = \sum_{i=1}^{2} w_i y_i \tag{21}$$

where y_1 and y_2 are peak overall IVR efficiency and area of inductor, respectively, with $w_1 = 5$ and $w_2 = -2$. Since the main focus of codesign is maximizing IVR efficiency, it has a higher weight compared with the inductor area. The black-box here uses the ten input parameters to compute the frequency response of the inductor using Ansys HFSS, which is then used to calculate the IVR efficiency.

Table I compares the performance of TSBO to nonlinear solver and IMGPO, along with a carefully hand-tuned design [39], [43]. Optimization using TSBO resulted in 85.1% peak efficiency for 5:1 V conversion with the inductor area of 5.16 mm^2 compared with 79.4% and 11.3 mm^2 with the hand-tuned design and 84.4% with 6.64 mm^2 for IMGPO. Though all algorithms started from the same initial point,

TABLE I
OPTIMIZATION RESULTS FOR IVR

	Hand Tuned	Non-Linear Solver	IMGPO	TSBO
Peak eff. 5V:1V	79.4%	78.6%	84.4%	85.1%
Area (mm^2)	11.3	25.2	6.64	5.16
Number of Simulations	N/A	>100	59	27
CPU Time (minutes)	Months	>185	115.6	50.1

TSBO converged after 51.1 min (27 simulations) compared with 115.6 min (59 simulations) for IMGPO. Optimization using nonlinear solver, the only non-ML algorithm, resulted in 78.6% efficiency with an inductor area of 25.2 mm^2, could not converge in 185 min (100 simulations) and was worse than the hand-tuned design.

The performance gap between TSBO and other ML and non-ML-based methods, both in terms of final value achieved and the convergence rate, shows that readily available BO methods may not be directly applicable to SI and PI problems.

C. Scaling to Higher Dimensionality: DPTBO

A problem with BO is that it does not scale well as the dimensionality increases. In the SI and PI domain, this occurs when all the parameters have both independent and joint effect (coupling) on $f(x)$, which causes the GP surrogate model to require lots more data to identify these effects.

To address high-dimensional problems, we can modify the GP model to explicitly search for these interactions. In DPTBO, we do this by rewriting the high-dimensional function $f(x)$ as a weighted sum of functions with lower order interactions, written as

$$f(x) = w_1 \sum_{i=1}^{D} f_i(x_i) + w_2 \sum_{1\le i<j\le D} f_{ij}(x_i, x_j) + \cdots + w_n \sum_{i_1<\cdots<i_n}^{D} f(x_{i_1}, \ldots, x_{i_n}) + w_D f(x_1, x_2, \ldots, x_D) \quad (22)$$

where the first summation describes the independent effect of each parameter (first order), the second summation describes the joint effect of any two parameters (second order), and so on, up to a maximum of n orders, where $n = 2$ or 3 usually captures the most important interactions. We preserve the highest order ($n = D$) to describe the remaining effects. The subfunctions in (22) describe the unknown nonlinear behavior for their corresponding order and each can be represented by a separate GP. Since training a separate GP for each order can be CPU intensive, we embed the entire weighted sum into the kernel of a single GP as described in [31] and shown in Fig. 28(a). This kernel-level weighted-sum approach is also called as an ADD-GP [44].

It is important to note here that weights corresponding to each order of interaction in (22) are learned during the training of ADD-GP. This allows to automatically adjust the contribution of different orders and dynamically strengthen or weaken the effect of different orders based on the collected

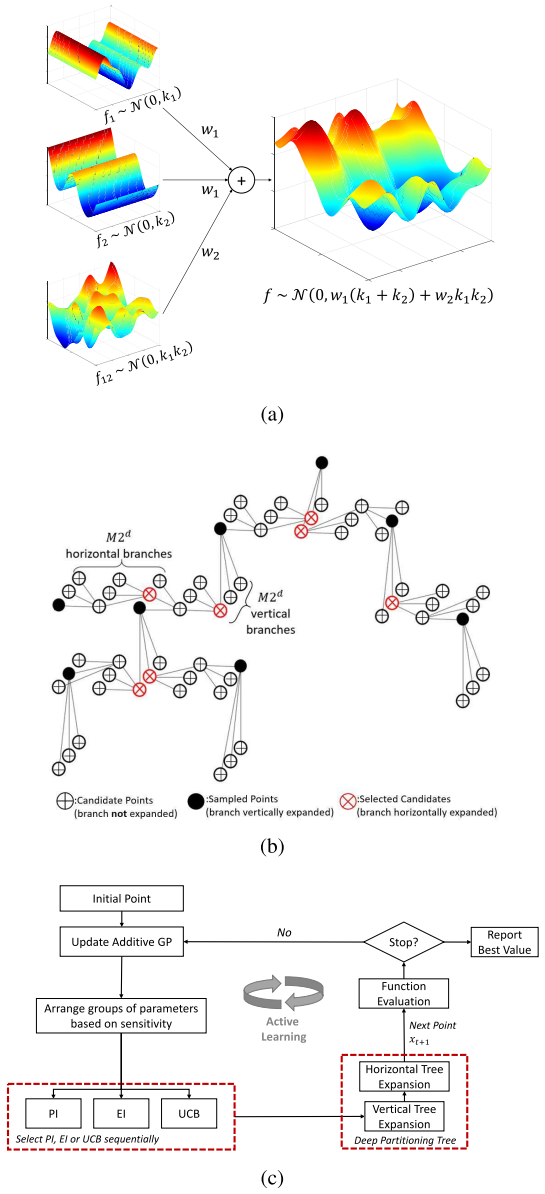


Fig. 28. Illustration of the DPTBO method [31]. (a) Lower order decomposition of a 2-D function. (b) Example DPT. (c) Flowchart.

data. For instance, the training can maximize w_D and minimize $w_{1,\ldots,n}$ for a particular problem to automatically discard the lower order effects, thereby recovering a regular GP or vice versa for other problems.

To cover large sample spaces, we consider the partitioning tree approach of TSBO as a sampling strategy. However, the approach is not directly applicable to high-dimensional problems. The subregions generated by the partitioning tree increases in volume with increasing dimensionality. Hence, to obtain a small enough region in high-dimensional spaces, we need to perform more partitions, which lead to more function queries (EM simulations) and slower convergence rates.

We, therefore, develop a new partitioning structure called deep partitioning tree (DPT), where the region division is guided by the sensitivity of parameters and occurs in two directions (vertical and horizontal) as opposed to a single

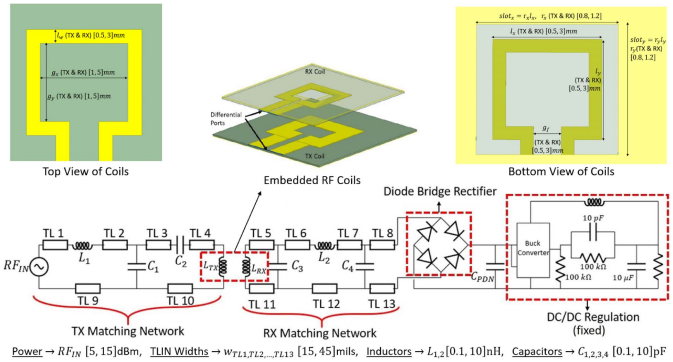


Fig. 29. WPT-based power delivery using embedded RF coils [31].

TABLE II
OPTIMIZATION RESULTS FOR WPT SYSTEM

	PSO	ADD-MES	DPT-BO
RX Coil Area (mm^2)	7.48	19.26	11.04
RF-DC Efficiency (%)	45.83	58.86	59.57
Number of Simulations	192	163	145
CPU Time (minutes)	691.2	603.1	536.5

direction (vertical) with TSBO. The main idea is that instead of performing 2^D region divisions, we group the parameters in M groups and perform a small scale partitioning to get $M2^d$ subregions where $D = Md$. As the small-scale partitioning leads to larger subregions, we determine the most promising subregion based on the acquisition function and iteratively shrink it by prioritizing the parameters with higher learned sensitivity. An example DPT is shown in Fig. 28(b). For brevity, we refer readers to [26] for a detailed description of the DPT method.

As for the acquisition function, we consider a library of strategies (EI, PoI, and UCB) similar to TSBO and select a different strategy at each iteration. Unlike TSBO, we do not select the best strategy as such selection will not be reliable in high-dimensional problems [31]. The overall flowchart of the DPTBO method is given in Fig. 28(c), and its open-source MATLAB implementation can be found at `https://github.com/GT-PRC/DPTBO`.

We now provide a 32-D design optimization example and compare DPTBO to a state-of-the-art high-dimensional BO method, maximum entropy search with additive GP (ADD-MES) [45], and a non-ML method, PSO.

1) Problem 7—WPT-Based Power Delivery for IoT: Consider optimization of a WPT-based power delivery solution for IoT devices, as shown in Fig. 29. The optimization goal is to determine 32 input parameters in Fig. 29 to maximize RF–dc power conversion efficiency at 1 GHz while minimizing the area of the RX coil. The multiobjective optimization is formulated as a weighted sum similar to the IVR example and RF–dc efficiency is prioritized over area. We use HFSS to characterize RF coils and Keysight ADS [46] to calculate the efficiency [31].

The optimization results are given in Table II [31]. Optimization using DPTBO resulted in 59.57% RF–dc conversion efficiency and RX coil area of 11.04 mm^2. It can be seen

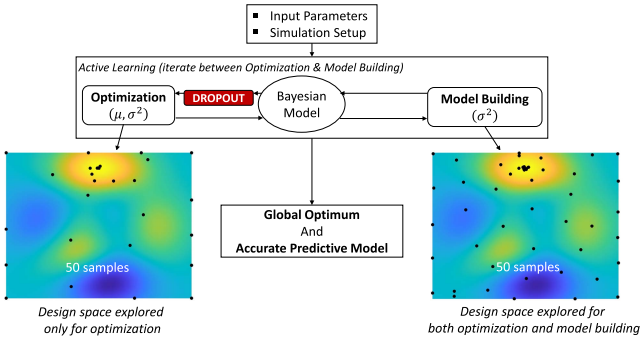


Fig. 30. Simultaneous optimization and model building using BALDO.

that DPTBO significantly outperforms ADD-MES in terms of RX coil area while getting a similar RF–dc efficiency and converging faster. Though PSO provided the lowest RX coil area, it performed significantly worse in terms of RF–dc efficiency (main goal) compared with ADD-MES and DPTBO.

IV. BAYESIAN ACTIVE LEARNING FOR UNCERTAINTY QUANTIFIED MODELS

As mentioned earlier, deterministic NNs covered in Section II assume that the predictions made are always accurate. This can be dangerous since uncertainty of the predictions is as important as the predictions themselves and should be accounted for in the model. We call this as uncertainty quantified model development, which is the subject of this section.

In particular, we present a Bayesian learning approach to perform well-calibrated uncertainty calculations that lead to reliable confidence intervals around the predictions. We then combine this with the active learning scheme presented in Section III to derive an uncertainty quantified model using the minimum amount of intelligently collected samples, which can then be used to obtain the predictions of interest and confidence intervals around them. These include, but are not limited to: 1) output PDF to estimate the effect of arbitrarily correlated process variations; 2) sensitivity of input parameters on the output response; and 3) worst case scenarios to ensure compliance. Since design respins occur due to noncompliance with specifications, item 3) becomes extremely critical for SI and PI and will be covered here. Details related to 1) and 2) are available in [47].

We present a technique called BALDO [47] that uses GPs to obtain these objectives, where we introduce the concept of active learning for simultaneous probabilistic model building [for item 1) and 2)] and optimization [for item 3)] to obtain all the predictions of interest with a single model in an automated fashion as shown in Fig. 30, as opposed to separate frameworks that require manual human intervention.

A. Bayesian Training of GPs for UQ

One of the challenges in ML is the fixing of parameters during the training process. For example, we trained the NN to obtain its weights and fixed them to obtain the NN predictions in Section II. Similarly, in Section III, we fixed the hyperparameters of the GP, θ, to the values found during training.

For developing the GP model, the fixed hyperparameter setting affects the quality of the confidence bounds that our surrogate provides since we constrain ourselves on the correctness of a single θ value through the training procedure. This means that the predictive uncertainty of the GP in (19) only accounts for data-related uncertainty and not parameter-related uncertainty. The data-related uncertainty here refers to a lack of data in specific regions of the sample space, which is sufficient to consider for optimization, but does not provide well-calibrated uncertainties.

To develop a reliable uncertainty quantified model, we need a more comprehensive GP model that makes fewer assumptions. Hence, the confidence bounds also need to account for parameter-related uncertainties. This means that instead of assuming a fixed θ, we need to consider all possible θ values and use a weighted sum of confidence intervals where the bounds obtained with more likely θ values affect the final confidence bound more than others. In the continuous θ domain, this corresponds to an integration that can be written as

$$p(y^* \mid x^*, D_t) = \int p(y^* \mid x^*, D_t, \theta) p(\theta \mid D_t) d\theta \quad (23)$$

where $D_t = (X_t, Y_t)$ is data we have at the tth iteration of the active learning. At a test point x^*, our model now predicts a distribution, $p(y^* \mid x^*, D_t)$, that no longer depends on θ and is a weighted sum of all possible distributions that we can get with a fixed hyperparameter, $p(y^* \mid x^*, D_t, \theta)$. We now need to learn the distribution $p(\theta \mid D_t)$ to use them as weights and compute the integral. Since this is analytically intractable, we resort to a technique called MCMC [48]. This learning procedure now becomes training for the GP. We refer readers to [48] for a detailed explanation of training the GP using the MCMC approach. Once the GP is trained, the predictions and confidence intervals that also accounts for parameter-related uncertainties can be obtained as in [47].

B. Sampling Strategy in BALDO

Active learning is equivalent to BO when the goal is to find the global optimum. When the goal is to build a probabilistic model with minimum amount of data, it is called BAL. Here, the next sampling point is selected to decrease the prediction uncertainty (entropy) of the model, i.e., the predicted variance of the GP in (18).

As described earlier, in SI and PI problems, we need an uncertainty quantified model that can also provide the worst case scenario (global minimum). Using BAL with entropy criterion aims to provide an accurate model but cannot identify the worst case scenario. On the other hand, using BO provides the worst case scenario but not an accurate model.

Instead of performing a BAL followed by BO or vice versa for a complete model, in BALDO, we introduce the concept of simultaneous model building and optimization. Here, the goal is to jointly derive an accurate predictive model over whole sample space while converging to the worst case scenario. To achieve this, we approach the active learning problem in two stages, namely optimization stage and learning stage, and we sequentially alternate between these at every iteration. The information obtained in the two stages is fused in a single GP model that is trained using the MCMC approach. To prioritize finding the worst case scenario due to its importance in SI and PI problems, we introduce a technique called dropout, as shown in Fig. 30. For brevity, we refer readers to [47] for details of the BALDO strategy. An open-source MATLAB implementation of BALDO can be found in `https://github.com/GT-PRC/BALDO`.

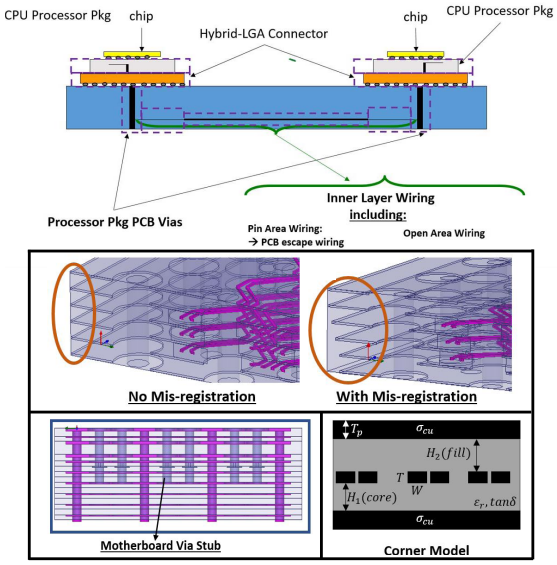


Fig. 31. Structure of IBM P9 processor to processor X-Bus channel [47].

TABLE III
UNCERTAIN PARAMETERS OF THE HIGH-SPEED CHANNEL

Parameters	Values
Motherboard Impedance Corner	Low, Nominal, High
Motherboard Attenuation Corner	Low, Nominal, High
Backdrilled Via Stub Length (mm)	0.101, 0.203, 0.305, 0.406, 0.508
Pin Area Wiring Mis-registration (mils)	0, 1.25, 2.5, 3.75, 5.00
TX CPU Package Corner Impedance	Low, Nominal, High
RX CPU package Corner Impedance	Low, Nominal, High

C. Problem 8—High-Speed Channel Signaling

We consider a comprehensive industrial example, IBM's POWER9 processor to processor X-Bus channel [49]. The topology of the channel is given in Fig. 31 and it operates at a data rate of 16 Gb/s using differential signaling. The objective here is to derive a complete uncertainty quantified GP model that can predict horizontal eye opening (HEYE) with confidence bounds while accurately identifying the worst case channel variable combination that gives the minimum HEYE. The problem is posed as a 6-D problem as in Table III [47]. We consider a discrete sample space to take advantage of precomputed S-parameters and corner cases.

The simulation framework used consists of three main parts and starts with generating 36-port S-parameters (one victim and eight differential aggressors) of end-to-end channels that contain bumps, CPU packages, and complete motherboard wiring. The channel response is then combined with behavioral models of TX and RX circuitry, where equalization settings

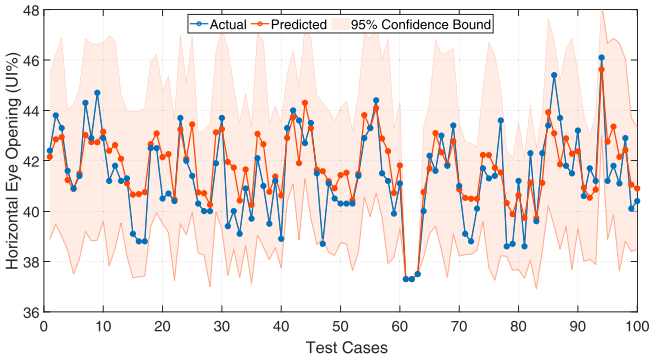


Fig. 32. Predicted HEYE and confidence intervals for 100 validation cases using the GP obtained with BALDO (average absolute error = 0.86%UI).

are determined by sweeping all possible combinations in time domain. The best setting is then used in a 10 million bit time-domain simulation to characterize the HEYE. Further details of the simulation setting can be found in [47]. It should be noted that although the sample space contains only 1125 different combinations, each simulation takes approximately 45 min, thereby simply sweeping that all combinations in a sequential manner can take up to 35 CPU days.

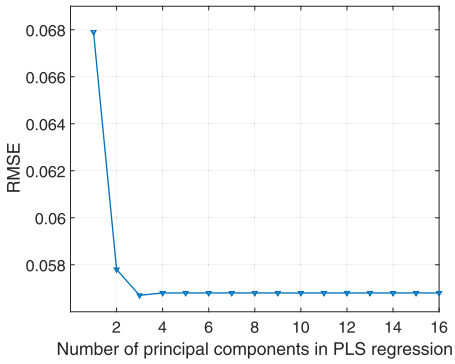
After running 50 simulations using the inputs determined by BALDO and the final GP model to predict all other channel combinations, we find that the average 95% confidence interval around the predictions to be ±3.1%UI, where UI is the unit interval used in eye diagrams. To show the quality of the confidence intervals, we collect 100 test samples and compare them against the predictions by the GP model in Fig. 32. It can be seen that the GP model correlates well with the actual simulations. The absolute error between the predicted eye widths and the ones obtained from actual simulations, averaged over every case in the test set, is calculated to be 0.86%UI.

More importantly, we observe that all the test cases are within the 95% confidence intervals around the predictions, showing the quality and reliability of the model. Furthermore, the model is observed to be confident and accurate to identify a worst case HEYE of 37.3%UI. In terms of run times, it took ∼37.6 h to collect the 50 simulations and <1 min to train the GP model. Once trained, the GP model can predict HEYE of all 1125 channel combinations and the confidence bounds in ∼0.7 s compared to 35 days using the aforementioned simulation framework.

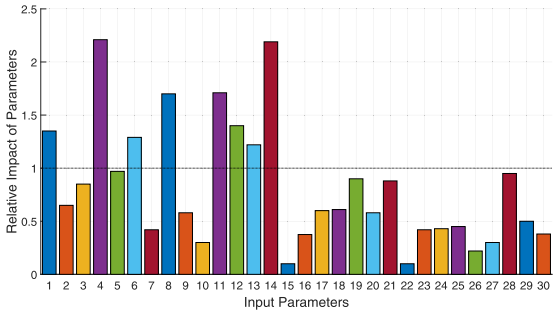
The confidence bounds around these predictions also provide informative feedback. If the model uncertainty is greater than an acceptable margin, more simulations can be performed to reduce the width of the confidence intervals, thereby answering the question of how many data samples are required to derive a reliable model. We refer readers to [47] for more results, including PDF of HEYE and sensitivity analysis with reliable confidence intervals, and performance comparison to GPs obtained using different approaches.

V. Future Directions and Opportunities

There is a clear trend in the semiconductor industry toward heterogeneous integration (HI) where large monolithic integrated chips partitioned into multiple chiplets will be connected together on an interconnect fabric along with chips from other processes, leading to complex high-density interposers. This poses a unique opportunity for use of ML not just in SI and PI but in other areas as well, without which designing such complex subsystems might become practically impossible. We discuss three possible areas for research here along with some initial ideas.



(a)



(b)

Fig. 33. PLS regression for WPT [50]. (a) Reduction in dimensionality. (b) Parameter sensitivities generated from the reduced model.

A. Addressing Higher Dimensionality

The methods presented in this article can handle dimensionalities up to $D \approx 50$. This is a large dimensionality in itself but may not be enough as we move forward to more advanced interposers. Part of the reason for such a limitation in dimensionality is the fidelity of the surrogate models as the number of parameters increases. It appears at this time that the only solution to this problem is to conduct a sensitivity analysis *a priori*, determine the most sensitive parameters, and use this as part of the ML process. However, this is a difficult problem since it is not easy to determine parameter sensitivities without lots of data, which can be challenging in itself if the data have to be generated from a 3-D EM solver.

One possible direction is by utilizing the PLS method, which is a technique that allows identification of relationships between a large number of input variables (that are highly correlated) and the response variable, using a small number of samples. The principle is based on projecting the input variables onto a new space defined by new variables, called principal components, which are linear combinations of the input variables.

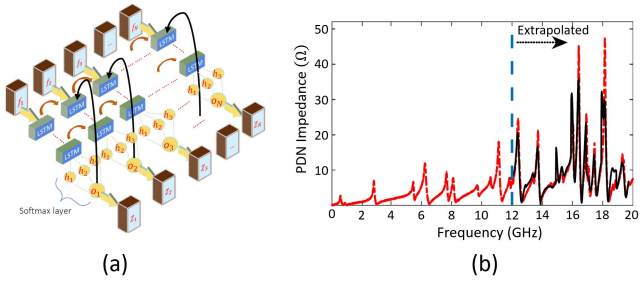


Fig. 34. (a) LSTM architecture. (b) Extrapolated PDN frequency response [51].

As an example, consider the WPT example in Fig. 29 used earlier with 30 parameters. Using 200 realizations from LHS, we develop a PLS regression model [50]. Fig. 33(a) shows the value of the root-mean-squared error (RMSE) computed with a ten-fold cross-validation procedure with respect to the number of principal components. The value of the RMSE significantly decreases from 1 to 3 principal components and then becomes quasi-constant from 4 to 16 principal components, indicating that three principal components are sufficient (reduction in dimensionality) to construct a surrogate model and to determine parameter sensitivities, as shown in Fig. 33(b). Though the surrogate model generated might not be the most accurate, it is sufficient to determine sensitivities for reducing dimensionality that can then feed into ML procedures that follow.

B. Extrapolation

ML models described in this article are developed based on a parameter range, and hence, the models are only valid within this range. An important question to ask is: "Can surrogate models be used to extrapolate outside the range over which it has been developed?" If this is possible, then models can be reused even if the parameter range is expanded. For packaging, this also addresses an important aspect related to resonances especially for PI analysis, where poles occurring close (but outside) to the parameter range can lead to a large increase in impedances, resulting in excessive power supply noise.

As an example, consider extrapolating the frequency response of a PDN beyond the range it has been simulated. We start by relying on the frequency samples being correlated in frequency space. This is expected since any PDN can be represented as a distributed array of transmission lines connected to passive elements. This means that knowing the value of the impedance at one resonant frequency determines implicitly the value at another frequency point. Using the information embedded in the band-limited space, we can then predict the poles and transitions at higher frequency by treating the response as a set of sequenced data. Learning this behavior is possible by using RNN discussed earlier, but by changing the architecture into an LSTM-RNN as in Fig. 34(a) [51]. In Fig. 34(b), the predicted extrapolated impedance response is shown for a PDN. Here, we set the cutoff frequency as 12 GHz beyond which is the extrapolated space. The predicted values follow the correct trend. The trained LSTM-RNN network learns which past values are important and retains them to estimate the next value of impedance. This means that poles have spatial correlation, and therefore, a future pole can be predicted from the past pole values without knowing the structure of the network explicitly. As shown in the figure, we are able to achieve a 66% extension in bandwidth with an MSE of 0.008. These early results, though promising, require further investigation.

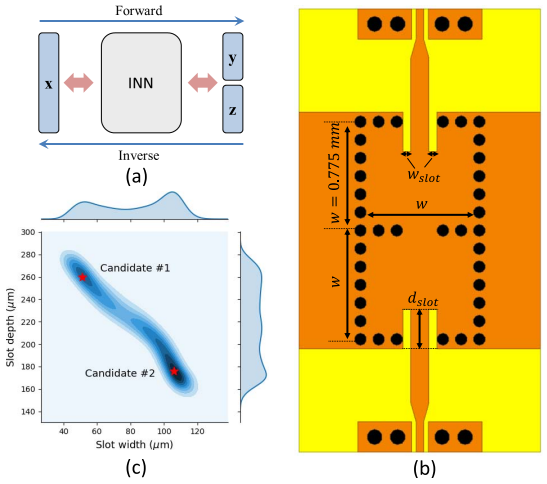


Fig. 35. Inverse design [52]. (a) INNs. (b) SIW with two parameters. (c) Probability distributions and candidate points.

C. Inverse Design

The ML procedures described learn the relationships between the input and output parameters through a model, using which the output response can be predicted given the input parameters. However, would not it be useful if we do the reverse, where the input parameters are predicted given the output response? This procedure is oftentimes used in filter design through design equations where based on output specifications, the circuit component values can be determined (also called synthesis). But can this be scaled where using a black-box approach (without design equations and only with data), we can do an inverse design. This can be very useful in DSE and design implementation. However, such a concept is an ill-posed problem, meaning that for a single output response, multiple combinations of inputs are possible. One possible method to address this is by introducing a latent variable space (z) represented using known distributions, using the latent space to learn the nonlinear transformation between the known distribution of the latent variables and the original data distribution [$p(x, y) \leftrightarrow p(z, y)$], and using this information to predict the conditional posterior (resulting) distribution of the input parameters given the output parameter $p(x|y)$, as shown in Fig. 35(a). This is possible using INNs.

As an example [52], consider a second-order SIW D-band filter as shown in Fig. 35(b) with two input design parameters, namely slot width (w_{slot}) and slot depth (d_{slot}). The target output parameters are center frequency (f_c) in passband and roll-off (slope of insertion loss). After INN training, the network generates the conditional probability distributions as shown in Fig. 35(c) for an output design target of

$f_c = 142$ GHz and roll-off = 2.6 dB/GHz. From the predicted distributions, two candidate designs are picked by selecting the peak distribution density points in the two regions, which are marked as stars in Fig. 35(c). Though both candidates meet the performance specifications, candidate 1 has a higher Q with better roll-off than candidate 2 when simulated using a 3-D EM solver. We believe that such INN-based approaches provide significant opportunities for designers to reduce design cycle time.

VI. Conclusion

ML has come a long way over the last 15 years since the first time it was applied to packaging. Over these years, the ML algorithms have become more robust, while the computing infrastructure has improved leaps and bounds through faster processors and cheaper memory. During this same period, new packaging technologies, such as SiP and system on package (SoP), have evolved, leading to significantly more integration in the package. With the trend toward HI, this appears to be the ideal intersection point where advanced ML techniques can be applied for enabling robust heterogeneously integrated future systems. We therefore strongly believe that there is an abundance of opportunities available moving forward where packaging can benefit immensely through the use of ML-based techniques not just in design, but in other areas as well.

In this article, we have addressed the design of FFNN, RNN, and CNN for various SI/PI problems where the focus was on fast model development. We then discussed the application of ML in the context of optimization using the BAL and deriving models that accommodate uncertainty, with an objective of removing the "human from the loop" during the design process. In its current state, we believe that ML-based design techniques can be an asset to any SI/PI engineer for improving productivity. With a wider adoption of such advanced techniques, there is an opportunity for further advancements in this area in the near future.

References

[1] B. Mutnury, M. Swaminathan, and J. P. Libous, "Macromodeling of nonlinear digital I/O drivers," *IEEE Trans. Adv. Packag.*, vol. 29, no. 1, pp. 102–113, Feb. 2006.

[2] Q.-J. Zhang, K. C. Gupta, and V. K. Devabhaktuni, "Artificial neural networks for RF and microwave design-from theory to practice," *IEEE Trans. Microw. Theory Techn.*, vol. 51, no. 4, pp. 1339–1350, Apr. 2003.

[3] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035. [Online]. Available: https://pytorch.org/

[4] *Deep Learning Toolbox*. Accessed: Jun. 14, 2020. [Online]. Available: https://www.mathworks.com/products/deep-learning.html

[5] W. T. Beyene, "Application of artificial neural networks to statistical analysis and nonlinear modeling of high-speed interconnect systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 1, pp. 166–176, Jan. 2007.

[6] Q.-J. Zhang and L. Zhang, "Neural network techniques for high-speed electronic component modeling," in *Proc. Int. Microw. Workshop Ser. Signal Integrity High-Speed Interconnects*, Feb. 2009, pp. 69–72.

[7] C. M. Schierholz, K. Scharff, and C. Schuster, "Evaluation of neural networks to predict target impedance violations of power delivery networks," in *Proc. IEEE 28th Conf. Electr. Perform. Electron. Packag. Syst. (EPEPS)*, Oct. 2019, pp. 1–3.

[8] T. Lu, J. Sun, K. Wu, and Z. Yang, "High-speed channel modeling with machine learning methods for signal integrity analysis," *IEEE Trans. Electromagn. Compat.*, vol. 60, no. 6, pp. 1957–1964, Dec. 2018.

[9] C. H. Goay, A. Abd Aziz, N. S. Ahmad, and P. Goh, "Eye diagram contour modeling using multilayer perceptron neural networks with adaptive sampling and feature selection," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 9, no. 12, pp. 2427–2441, Dec. 2019.

[10] S. Chen, J. Chen, T. Zhang, and S. Wei, "Semi-supervised learning based on hybrid neural network for the signal integrity analysis," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, early access, Oct. 21, 2019, doi: 10.1109/TCSII.2019.2948527.

[11] Y. Liu, T. Lu, J. Y. Kim, K. Wu, and J.-M. Jin, "Fast and accurate current prediction in packages using neural networks," in *Proc. IEEE Int. Symp. Electromagn. Compat., Signal Power Integrity (EMC+SIPI)*, Jul. 2019, pp. 621–624.

[12] H. Yu, H. Chalamalasetty, and M. Swaminathan, "Modeling of voltage-controlled oscillators including I/O behavior using augmented neural networks," *IEEE Access*, vol. 7, pp. 38973–38982, 2019.

[13] H. Yu, M. Swaminathan, C. Ji, and D. White, "A nonlinear behavioral modeling approach for voltage-controlled oscillators using augmented neural networks," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2018, pp. 551–554.

[14] CADENCE. *Cadence Spectre*. Accessed: Jun. 14, 2020. [Online]. Available: http://www.cadence.com

[15] B. Mutnury, M. Swaminathan, M. Cases, N. Pham, D. de Araujo, and E. Matoglu, "Macro-modeling of non-linear pre-emphasis differential driver circuits," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2005, pp. 1–4.

[16] Q. J. Zhang, Y. Cao, and I. Erdin, "Fast IO buffer modeling using neural network methods," in *Proc. 11th Int. Conf. Electron. Packag. Technol. High Density Packag.*, Aug. 2010, pp. 666–669.

[17] T. Nguyen, T. Lu, J. Sun, Q. Le, K. We, and J. Schut-Aine, "Transient simulation for high-speed channels with recurrent neural network," in *Proc. IEEE 27th Conf. Electr. Perform. Electron. Packag. Syst. (EPEPS)*, Oct. 2018, pp. 303–305.

[18] 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 *Proc. IEEE 69th Electron. Compon. Technol. Conf. (ECTC)*, May 2019, pp. 1889–1896.

[19] G. Signorini, C. Siviero, S. Grivet-Talocia, and I. S. Stievano, "Macromodeling of I/O buffers via compressed tensor representations and rational approximations," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 6, no. 10, pp. 1522–1534, Oct. 2016.

[20] H. Yu, T. Michalka, M. Larbi, and M. Swaminathan, "Behavioral modeling of tunable I/O drivers with preemphasis including power supply noise," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 233–242, Jan. 2020.

[21] H. Park *et al.*, "Deep reinforcement learning-based optimal decoupling capacitor design method for silicon interposer-based 2.5-D/3-D ICs," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 10, no. 3, pp. 467–478, Mar. 2020.

[22] H. Ma, E.-P. Li, J. Schutt-Aine, and A. C. Cangellaris, "Deep learning method for prediction of planar radiating sources from near-field intensity data," in *Proc. IEEE Int. Symp. Electromagn. Compat., Signal Power Integrity (EMC+SIPI)*, Jul. 2019, pp. 610–615.

[23] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016, *arXiv:1603.07285*. [Online]. Available: http://arxiv.org/abs/1603.07285

[24] H. M. Torun *et al.*, "A spectral convolutional net for co-optimization of integrated voltage regulators and embedded inductors," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[25] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[26] H. M. Torun, A. C. Durgun, K. Aygun, and M. Swaminathan, "Enforcing causality and passivity of neural network models of broadband S-parameters," in *Proc. IEEE 28th Conf. Electr. Perform. Electron. Packag. Syst. (EPEPS)*, Oct. 2019, pp. 1–3.

[27] ANSYS. *Ansys HFSS ver. 2019.1*. Accessed: Jun. 14, 2020. [Online]. Available: http://www.ansys.com

[28] H. M. Torun, A. C. Durgun, K. Aygun, and M. Swaminathan, "Causal and passive parameterization of s-parameters using neural networks," *IEEE Trans. Microw. Theory Techn.*, 2020, doi: 10.1109/TMTT.2020.3011449.

[29] J. Gonzalez, "Introduction to Bayesian optimization," presented at the Gaussian Process Summer School, Sheffield, U.K., Sep. 2017. [Online]. Available: http://gpss.cc/gpss17/slides/gpss_bayesopt2017.pdf

[30] T. Dhaene, "Challenges in the optimization of modern RF and em systems: A Bayesian perspective," in *Proc. IEEE Electr. Design Adv. Packag. Syst. Symp. (EDAPS)*, Dec. 2018.

[31] H. M. Torun and M. Swaminathan, "High-dimensional global optimization method for high-frequency electronic design," *IEEE Trans. Microw. Theory Techn.*, vol. 67, no. 6, pp. 2128–2142, Jun. 2019.
[32] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2006. [Online]. Available: http://www.gaussianprocess.org/gpml/
[33] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn.*, 2010, pp. 1015–1022.
[34] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.
[35] Z. Kiguradze, J. He, B. Mutnury, A. Chada, and J. Drewniak, "Bayesian optimization for stack-up design," in *Proc. IEEE Int. Symp. Electromagn. Compat., Signal Power Integrity (EMC+SIPI)*, Jul. 2019, pp. 629–634.
[36] M. A. Dolatsara and M. Swaminathan, "Determining worst-case eye height in low BER channels using Bayesian optimization," in *Proc. IEEE 11th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2020, pp. 1–4.
[37] R. Medico, D. Spina, D. Vande Ginste, D. Deschrijver, and T. Dhaene, "Machine-learning-based error detection and design optimization in signal integrity applications," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 9, no. 9, pp. 1712–1720, Sep. 2019.
[38] S. De Ridder, D. Spina, N. Toscani, F. Grassi, D. V. Ginste, and T. Dhaene, "Machine-learning-based hybrid random-fuzzy uncertainty quantification for EMC and Si assessment," *IEEE Trans. Electromagn. Compat.*, early access, Apr. 17, 2020, doi: 10.1109/TEMC.2020.2980790.
[39] H. M. Torun, M. Swaminathan, A. Kavungal Davis, and M. L. F. Bellaredj, "A global Bayesian optimization algorithm and its application to integrated system design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 4, pp. 792–802, Apr. 2018.
[40] M. Hoffman, E. Brochu, and N. de Freitas, "Portfolio allocation for Bayesian optimization," in *Proc. 27th Conf. Uncertainty Artif. Intell.*, 2011, pp. 327–336.
[41] J. Xie and M. Swaminathan, "Electrical-thermal co-simulation of 3D integrated systems with micro-fluidic cooling and joule heating effects," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 1, no. 2, pp. 234–246, Feb. 2011.
[42] S. J. Park, B. Bae, J. Kim, and M. Swaminathan, "Application of machine learning for optimization of 3-D integrated circuits and systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 6, pp. 1856–1865, Jun. 2017.
[43] S. Müeller *et al.*, "Design of high efficiency integrated voltage regulators with embedded magnetic core inductors," in *Proc. IEEE 66th Electron. Compon. Technol. Conf. (ECTC)*, May 2016, pp. 566–573.
[44] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen, "Additive Gaussian processes," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 226–234.
[45] Z. Wang and S. Jegelka, "Max-value entropy search for efficient Bayesian optimization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 1–12.
[46] KEYSIGHT. *Keysight Ads ver. 2017*. Accessed: Jun. 14, 2020. [Online]. Available: http://www.keysight.com
[47] H. M. Torun, J. A. Hejase, J. Tang, W. D. Beckert, and M. Swaminathan, "Bayesian active learning for uncertainty quantification of high speed channel signaling," in *Proc. IEEE 27th Conf. Electr. Perform. Electron. Packag. Syst. (EPEPS)*, Oct. 2018, pp. 311–313.
[48] R. M. Neal, "Slice sampling," *Ann. Statist.*, vol. 31, no. 3, pp. 705–741, 2003.
[49] S. Chun *et al.*, "IBM POWER9 package technology and design," *IBM J. Res. Develop.*, vol. 62, nos. 4–5, p. 12, Jul./Sep. 2018.
[50] M. Larbi, H. M. Torun, and M. Swaminathan, "Estimation of parameter variability for high dimensional microwave problems via partial least squares," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2019, pp. 940–943.
[51] O. W. Bhatti and M. Swaminathan, "Impedance response extrapolation of power delivery networks using recurrent neural networks," in *Proc. IEEE 28th Conf. Electr. Perform. Electron. Packag. Syst. (EPEPS)*, Oct. 2019, pp. 1–3.
[52] H. Yu, H. M. T. Mutee, U. Rehman, and M. Swaminathan, "Design of siw filters in d-band using invertible neural nets," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Aug. 2020.

Madhavan Swaminathan (Fellow, IEEE) received the B.E. degree from the Regional Engineering College at Tiruchirapalli (now NITT), Tiruchirapalli, India, in 1985, and the M.S./Ph.D. degrees in electrical engineering from Syracuse University, Syracuse, NY, USA, in 1989 and 1991, respectively.

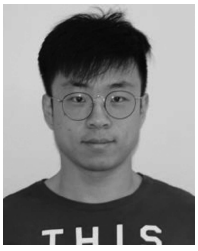
He is currently the John Pippin Chair in Microsystems Packaging and Electromagnetics with the School of Electrical and Computer Engineering (ECE), a Professor of ECE with a joint appointment in the School of Materials Science and Engineering (MSE), and the Director of the 3D Systems Packaging Research Center (PRC), Georgia Tech (GT), Atlanta, GA, USA. He also serves as the Site Director for the NSF Center for Advanced Electronics through Machine Learning (CAEML) and the Theme Leader for Heterogeneous Integration, SRC JUMP ASCENT Center. He formerly held the position of Founding Director at the Center for Co-Design of Chip, Package, System (C3PS), the Joseph M. Pettit Professor in Electronics in ECE, and the Deputy Director of the Packaging Research Center (NSF ERC), GT. Prior to joining GT, he was with IBM, working on packaging for supercomputers. He is the author of more than 500 refereed technical publications, holds 32 patents, primary author and co-editor of three books, the founder and co-founder of two start-up companies, and the founder of the IEEE Conference Electrical Design of Advanced Packaging and Systems (EDAPS), a premier conference sponsored by the EPS Society.

Dr. Swaminathan has served as the Distinguished Lecturer for the IEEE EMC Society.

Hakki Mert Torun (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2016, and the M.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2019, where he is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering.

He has coauthored more than 30 refereed technical publications. His current research interests include developing machine learning algorithms for system-level design optimization and modeling with the applications in signal and power integrity in high-speed channels, microwave electronics, and VLSI systems.

Mr. Torun was a recipient of the 2019 Georgia Tech ECE Graduate Student Excellence Award and the Best Student Paper Award of the IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS) in 2018.



Huan Yu received the B.S. degree in electrical and computer engineering from Zhejiang University, Hangzhou, China, in 2014. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the Georgia Institute of Technology, Atlanta, GA, USA.

His research interests include electronics modeling, simulation, and optimization, with a focus on machine learning.

Jose Ale Hejase (Member, IEEE) received the B.S. degree *(cum laude)* in electrical engineering from Oakland University, Rochester Hills, MI, USA, in 2006, and the M.S. and Ph.D. degrees in electrical and computer engineering—specializing in applied electromagnetics—from Michigan State University, East Lansing, MI, USA, in 2009 and 2012, respectively.

Since 2012, he has been with IBM Corporation Austin, TX, USA. He is currently a Senior Engineer at IBM, where he is working with the POWER servers hardware development organization. His responsibilities revolve around signal integrity design for the latest and highest performance high-speed computer sever bus links. His research interests include electromagnetic material characterization, electromagnetic nondestructive evaluation, millimeter-wave guiding and high-speed link channel design, and modeling and signal integrity analysis.

Dr. Hejase is a member of the Technical Program Committee of the IEEE EPEPS and IEEE SPI conferences and a member of the IEEE EPS technical committee on Electrical Modeling, Design and Simulation. He received the IBM Early Tenure Inventor Award in 2013, the IBM Outstanding Technical Achievement Awards in 2017 and 2019 for his work on developing POWER9 servers high-speed buses, and his fourth IBM Invention Plateau in 2020.

Wiren Dale Becker (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the University of Illinois at Urbana–Champaign, Champaign, IL, USA.

He is currently a Chief Engineer of electronic packaging integration with IBM Systems, Poughkeepsie, NY, USA. His responsibility is the electrical system packaging architecture of IBM Systems, including the design of high-speed channels to enable the computer system performance and the power distribution networks for reliable operation of the integrated circuits that make up the processor subsystem.

Dr. Becker has chaired the IEEE EPEPS Conference and the SIPI Embedded Conference of the EMC Symposium. He also chairs the IEEE EPS Technical Committee on Electrical Modeling, Design, and Simulation. He co-chairs the High-Performance Computing TWG of the HIR Roadmap. From 2017 to 2018, he was the IAB Chair for NSF U/ICRC Center on Advancing Electronics through Machine Learning (CAEML). He has chaired the iNEMI PEG on High-End Systems. including the chapter on the High-End Systems Roadmap from 2007 to 2017. He is a iNEMI Technical Committee Member and a member of SWE. He is a Senior Area Editor for the IEEE TRANSACTIONS ON COMPONENTS, PACKAGING, AND MANUFACTURING TECHNOLOGY.