IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

IEEE ELECTRONICS PACKAGING SOCIETY SECTION

Received August 10, 2021, accepted September 7, 2021, date of publication September 13, 2021,
date of current version September 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3112134

Transient Simulations of High-Speed Channels Using CNN-LSTM With an Adaptive Successive Halving Algorithm for Automated Hyperparameter Optimizations

CHAN HONG GOAY, NUR SYAZREEN AHMAD, (Member, IEEE), AND PATRICK GOH
School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Nibong Tebal, Penang 14300, Malaysia

Corresponding author: Patrick Goh (eepatrick@usm.my)

This work was supported by the Ministry of Higher Education, Malaysia, through the Fundamental Research Grant Scheme (FRGS) under Grant FRGS/1/2020/TK0/USM/02/7.

ABSTRACT Transient simulations of high-speed channels can be very time intensive. Recurrent neural network (RNN) based methods can be used to speed up the process by training a RNN model on a relatively short bit sequence, and then using a multi-steps rolling forecast method to predict subsequent bits. However, the performance of the RNN model is highly affected by its hyperparameters. We propose an algorithm named adaptive successive halving automated hyperparameter optimization (ASH-HPO) which combines successive halving, Bayesian optimization (BO), and progressive sampling to tune the hyperparameters of the RNN models. Modifications are proposed to the successive halving and progressive sampling algorithms for better efficiency on time series data. The ASH-HPO algorithm trains on smaller dataset subsets initially, then expands the training dataset progressively and adaptively adds or removes models along the process. In this paper, we use the ASH-HPO algorithm to optimize the hyperparameters of convolutional neural networks (CNNs), long short-term memory (LSTM) networks, and CNN-LSTM networks. We demonstrate the effectiveness of the ASH-HPO algorithm using a PCIe Gen 2 channel, a PCIe Gen 5 channel, and a PAM4 differential channel. We also investigate the effects of several settings and tunable variables of the ASH-HPO algorithm on its convergence speed. As a benchmark, we compared the ASH-HPO algorithm to three state-of-the-art HPO methods: BO, successive halving, and hyperband. The results show that the ASH-HPO algorithm converges faster than the other HPO methods on transient simulation problems.

INDEX TERMS Automated hyperparameter optimization, convolutional neural network (CNN), high-speed channel, long short-term memory (LSTM) network, progressive sampling, transient simulation.

I. INTRODUCTION

The fast and accurate transient simulations of electrical interconnects consisting of high-speed channels play an important role in the design and verification of electrical devices and circuits [1]–[3]. With the continuous increase in data throughput and operating frequency, interconnect signal integrity becomes an important consideration in order to meet the stringent timing and voltage requirements in modern systems. Thus, the modeling and simulations for signal integrity has become a topic which has garnered increasing attention over the last decade.

The associate editor coordinating the review of this manuscript and approving it for publication was Wiren Becker.

In the analysis of high-speed systems, an eye diagram contains information that allows the engineers to evaluate key performance metrics for signal integrity such as the amount of intersymbol interference, noise margin, timing jitter, and timing sensitivity. Traditionally, an eye diagram is generated by overlaying the voltage waveform obtained from a transient simulation using a circuit simulator [4]. However, this is a computationally expensive process especially when the transient simulation is performed over long bit sequences on highly non-linear and complex channels. Several methods such as the peak distortion analysis (PDA) [5], statistical eye (StatEye) analysis [6], and edge response analysis [7] are developed to perform fast eye diagram analysis for linear time-invariant (LTI) systems without generating the

long voltage waveforms. However, these methods are only applicable to LTI systems while the behavior of high-speed channels can be non-linear due to the presence of non-linear components such as the I/O drivers and receivers.

Recently, more modern transient simulation and eye diagram modeling techniques have been pursued, owing to the advancements in surrogate modeling and machine learning techniques. The main advantage of these modeling techniques is that the model, once developed, can be reused for future computational savings, because the prediction speed of the model is much faster than the training speed. For example, [8] uses a short transient simulation to train a polynomial chaos surrogate model and the surrogate model is then used to estimate the jitter and eye diagram of the output signal, while [9] uses Bayesian optimization to perform a worst-case analysis of the eye diagram. In addition, neural network based techniques have also been developed such as in the application of multilayer perceptron (MLP) neural network for the modeling of the eye diagram [10]–[13], jitter [14], channel equalization [15], and physical parameters such as resistance, conductance, inductance, and capacitance [16]. However, MLP lacks the ability to capture sequential information. For these applications, the recurrent neural network (RNN) is often used instead. RNN is a type of ANN that specializes in modeling time series data but it has also been used in the field of circuit design, especially in the modeling of non-linear devices such as CMOS receivers [17], power amplifiers [18]–[21], RF mixers [22], rectangular waveguides, and microstrip filters [23]. In the area of transient modeling, [24], [25] apply the long short-term memory (LSTM) network, a type of RNN, to perform fast transient simulations of high-speed channels. The LSTM network is trained on a short initial bit sequence, and it is then used to predict up to hundreds of thousands of bits. It is shown that the LSTM network outperforms a RNN in terms of the convergence rate and accuracy when both of them have the same memory length in various voltage waveform modeling tasks.

Although recent results have shown the potential of machine learning methods versus their traditional simulation counterparts, the amount of time spent for designing the networks and tuning its hyperparameters are often overlooked. A hyperparameter is a configuration variable whose value cannot be determined from training and must be set before training. The performance of a neural network does not only depend on the quality of the training data, but on its hyperparameter choices as well, and a bad set of hyperparameters can affect the accuracy of the neural network. The tuning process of these hyperparameters can be time consuming and often relies on expert engineering knowledge and know-hows. Thus, automated hyperparameter optimization (HPO) techniques have been developed, which aim to find the optimal hyperparameter choices based on the architecture, regularization, and optimization of the neural networks. More information on the field of HPO can be found in [26].

In this research, various types of HPO methods including the Bayesian optimization (BO), successive halving, and hyperband are studied. BO is one of the most popular HPO strategy and has been successfully applied to the field of signal integrity modeling [27]. The BO algorithm requires a surrogate model of the objective function, and each evaluation of the objective function usually involves training a neural network and computing its validation loss. Although BO has a good reputation of requiring only a small amount of objective function evaluations, each evaluation of the objective function can still be computationally expensive for large dataset or complex ANN models, since each evaluation involves training a model to completion. This also means that BO will waste some resources on bad configurations as it relies entirely on the surrogate models to avoid creating bad models in the first place. Bandit-based methods such as successive halving [28] and hyperband [29] avoid this problem by evaluating the configurations using partially trained models. Successive halving is a simple yet powerful HPO method where the initial budget is split evenly across all configurations, and successively, the budget for the half that performs worse is removed while the budget for the other half is doubled until only one configuration is left. Thus, resource spent on bad configurations can be reduced since these configurations will be removed in the early iterations. Other than that, some researchers have also proposed methods to evaluate the configurations based on smaller dataset subsets. For example, [30] proposes a progressive sampling-based BO algorithm which uses only a small subset of data for the objective function evaluation during the initial rounds to drop unpromising configurations, and allocates more resources on the promising configurations with larger datasets. However, the generation of the dataset subsets for the progressive sampling-based BO algorithm involves a random sampling strategy, which will destroy the sequence of a time series data. This makes it unsuitable in transient simulation modeling where it is important to preserve the original sequence of the values.

In this paper, we propose a HPO framework named adaptive successive halving HPO (ASH-HPO) that combines BO and the successive halving algorithm, while adapting the progressive sampling method and modifying it to preserve the original sequence of the time series data. The ASH-HPO algorithm is developed to perform the HPO task efficiently for time-series forecasting sequential models. Instead of sampling randomly, the progressive sampling is modified to sample along the data sequence starting from the first time step, and remove part of the training data from the initial time steps when the size of the training data gets too large. We apply the proposed ASH-HPO algorithm to perform HPO for the CNN, LSTM, and CNN-LSTM networks. The CNN-LSTM architecture involves using convolutional neural network (CNN) layers for feature extraction and LSTM layers for sequence prediction [31] and it has been reported that CNN-LSTM outperforms LSTM in various time series prediction tasks such as household energy consumption

prediction [32], visual recognition and description [33], rainfall prediction [34], and air pollution forecasting [35]. We demonstrate the robustness of the ASH-HPO algorithm using various problems for transient simulation modeling and compare its performance to the BO, successive halving and hyperband algorithms. Moreover, we also present the use of a multi-step rolling forecast method to reduce the prediction time of our models.

II. HYPERPARAMETER OPTIMIZATION (HPO) METHODS

The HPO process is a process of selecting a set of optimal hyperparameters, H within the search space, $\mathcal{S}$ for a machine learning algorithm. Grid search is the most basic and traditional HPO method. However, it is only applicable when the number of hyperparameters in $\mathcal{S}$ is low as it suffers from the curse of dimensionality, where the number of trials required to fully explore $\mathcal{S}$ grows exponentially with the number of hyperparameters. For a search space with seven hyperparameters, each with three possible values, the total required number of trials is then $3^7 = 2187$ which is a prohibitively large number. Random search can explore $\mathcal{S}$ much more effectively than grid search and usually outperforms the latter because grid search can sometimes allocate too many trials on unimportant hyperparameters [26]. Both methods are completely ignorant of their pass trials, which means that their past decisions do not help them make better decisions in the future. This causes the overspending of resources on evaluating bad hyperparameters. Thus, they are often replaced with more advanced HPO methods such as BO, successive halving, and hyperband.

A. BAYESIAN OPTIMIZATION (BO)

BO is very suitable for optimizing expensive objective functions, which makes it a very good candidate for HPO of deep neural networks. The steps in performing Bayesian optimization on a black box function are: (step 1) initialize the process by sampling the hyperparameter space to generate a small number of observations; (step 2) create a surrogate model by fitting it to the observations generated previously; (step 3) use the maximum value of the acquisition function to select the next point to sample in $\mathcal{S}$, the acquisition function balances exploitation and exploration where the acquisition value of a point is high when the predicted value from the surrogate model is high and the uncertainty about the predicted value is high; (step 4) evaluate the objective function at the newly sampled point and add it to the observations. Steps 2 to 4 will be repeated until the goal is met. Tutorials on Bayesian optimization can be found in [36]. In our work, BO is used to generate models in the first iteration of successive halving, and to generate additional models later in randomly selected iterations. Various acquisition functions are investigated including expected improvement (EI), entropy search, upper confidence bound (UCB), and probability improvement (PI) [37].

B. SUCCESSIVE HALVING

Successive halving is conceptually very simple. For a fixed budget, B and number of initial configurations, n, allocate the budget evenly across every configuration so that every configuration gets B/n resources. The budget refers to the epochs in this work, but can also refer to various types of resources such as the size of training data, the training time, and the number of features. At the end of each iteration, remove half of the configurations that performed worst, and double the resource of every surviving configurations and train them again. The process repeats until only one configuration is left. However, it is unclear whether to use larger or smaller n. When n is small, each configuration is given more resources, and that can be problematic when the resources are wasted on the bad configurations. When n is large, each configuration is given less resources, and this can also cause problems when the trainings are terminated before the potentially good configurations with slow convergence rates start to reveal themselves. The successive halving algorithm can be tuned to remove more models in every iteration by setting a higher value of the elimination factor, λ which determines the proportion of configurations eliminated in each iteration. Generally, approximately $(\lambda - 1)/\lambda$ of the models will be removed in each iteration and the budget of every surviving configuration will be increased by a factor of λ.

Despite these weaknesses, successive halving is very suitable in the training of time series data. The time series dataset can be broken down into several dataset subsets and the original sequencing of dataset is preserved by using the first subset in the first successive halving iteration, the second subset in the second iteration, and so on. The data are arranged in such a way that the last surviving configuration learns the full sequence, and other models only need to learn part of the sequence in each iteration of the successive halving process to reduce the training time per epoch. In our work, we apply the successive halving method and propose several modifications so that it is more efficient on time series problems.

C. HYPERBAND

Successive halving is based on an assumption that the best configurations are more likely to be in the top half than the bottom half of the configurations after a small number of training iterations. However, there are some exceptions to that. For example, when the hyperparameter to be optimized is the learning rate, a model with a smaller learning rate may lag behind the rest of the pack when the number of training iterations is small, and still outperform the rest of the configurations when the number of training iterations is large.

In order to combat this problem and the problem of the B/n trade off, many researchers have switched their attention to the hyperband algorithm [29]. Hyperband is a hedging strategy that divides the whole HPO problem into several

successive halving problems named brackets, each exploring a unique value of n for a fixed B. The user needs to define two inputs to the hyperband algorithm, R and λ. R is the maximum resource that can be allocated to a single configuration in any iteration, and λ is the elimination factor. In this work, we compare the ASH-HPO algorithm to the hyperband algorithm using three applications.

III. ADAPTIVE SUCCESSIVE HALVING AUTOMATED HYPERPARAMETER OPTIMIZATION (ASH-HPO)

In this section, our proposed ASH-HPO algorithm will be explained in detail. Subsection IV.A explains the method for converting the time series data to a supervised learning problem. Subsection IV.B explains the method for training the neural models. Subsection IV.C explains the role of BO in the ASH-HPO algorithm. Subsection IV.D explains the proposed modifications on the successive halving algorithm and the integration of it in the ASH-HPO algorithm. Subsection IV.E explains the method for retraining the bad performing models. Subsection IV.F explains the method for expanding the search space, $\mathcal{S}$. Subsection IV.G introduces penalties to eliminate the slow training models and bad performing models. Finally, subsection IV.H summarizes the overall workflow of the ASH-HPO algorithm.

A. PROGRESSIVE SAMPLING FOR TIME SERIES DATA

This subsection describes how the data are processed into dataset subsets for the ASH-HPO algorithm to make it more efficient for time series predictions. The accuracy of the surrogate model is important for any type of surrogate model based HPO framework because a good surrogate model can have a better chance of suggesting a set of good hyperparameters. The more trials we can afford, the more accurate our surrogate model is because it will have more past experience to learn from. However, each trial is usually computationally expensive when it involves training a model from scratch on a large dataset. Thus, a progressive sampling strategy is used so that we start the HPO process with a small training set, and its size is increased slowly as the process continues. However, training the models on a small subset of data can downgrade the quality of the surrogate model. Thus, new models will be generated using the BO algorithm, not only at the beginning when the data size is minimal, but also when the data size becomes larger. This way, there is a chance to correct the mistakes made by the surrogate model at the beginning of the algorithm.

In this work, a sliding window transformation method with a window length or lookback, W_{LB} is used to turn a time series dataset into a supervised learning problem. Through this process, the original time series data, D_0 will be normalized within the range of [0, 1] and transformed into another dataset named D with length t_N using the sliding window transformation method. Also, we name the elements in D as d, where d_t represents the data of the tth time step such that $D = \{d_0, d_1, d_2, \ldots, d_t, \ldots, d_{t_N}\}$. A bracket represents an iteration in the progressive sampling process and the training and validation samples are different for each bracket. We would like to introduce a few terms here. $D_t(i)$ and $D_v(i)$ are the training and validation data in the ith bracket, where $D_t(i), D_v(i) \subset D$. L_{t0} and L_{v0} are a unit size of training and validation samples, $L_t(i)$ is the size of the training samples in the ith bracket, and L_{max} is the maximum allowable size of the training samples, where L_{max} must be divisible by L_{t0}. In this work, the size of $D_v(i)$ is fixed to one unit or $(1 \times L_{v0})$ regardless of the bracket number. The size of $D_t(i)$ is L_{t0} in the first bracket, and its size is increased by one unit or $(1 \times L_{t0})$ each time the bracket number increases until its size reaches L_{max}. After that, the training samples from the earlier time steps will be removed so that L_t does not exceed L_{max}.

Fig. 1 illustrates the sliding window process. As can be seen from the figure, $(1 \times L_{t0})$ of earlier time steps are removed from D_t in bracket 4, $(2 \times L_{t0})$ of earlier time steps are removed from D_t in bracket 5 and so on. This figure also visualizes the arrangements of $D_t(i)$ and $D_v(i)$ with the assumption that there is no new model created after the first bracket. However, if there is a new model created in any other bracket, the training data will always start at the beginning of all time steps even if the $L_t(i)$ will exceed L_{max}. This is to ensure that every new model can learn from the exact same dataset as the old models without any missing information. This method for arranging D_t and D_v is very similar to that of the blocked cross validation. Since all the D_v must be placed after the D_t to prevent data leakage from the future, data arrangement methods that splits D_t and D_v randomly, such as the regular k-fold cross validation method is not suitable for time series data. Fig. 2 shows a flowchart for generating $D_t(i)$ and $D_v(i)$ and the function is named GenerateTrainingAndValidationData(). $\Re$ is a random positive integer between two and six which decides in which bracket to generate new models by utilizing the BO algorithm, t_a and t_b are the beginning and the end of the training data sequence where $L_t(i) = (t_b - t_a)$, t_c and t_d are the beginning and the end of the validation data sequence where $L_v(i) = L_{v0} = (t_d - t_c)$. There is a special case when the counter is equal to $\Re$, where new models will be generated through BO and t_a will be set to one automatically so that the newly generated model can learn from the beginning of the dataset. We compute t_a, t_b, t_c, and t_d as follows:

$$t_a = \begin{cases} 1, & \text{Counter} = \Re \\ 1, & i \times L_{t0} \leq L_{\max} \\ \left(i - \dfrac{L_{\max}}{L_{t0}}\right) \times L_{t0} + 1, & i \times L_{t0} > L_{\max} \end{cases} \quad (1)$$

$$t_b = (i+1) \times L_{t0} \quad (2)$$

$$t_c = t_b + 1 \quad (3)$$

$$t_d = t_c + L_{v0}. \quad (4)$$

B. TRAINING THE MODELS

In this subsection, the method used to train the models with a minimal amount of budget will be explained. In order to keep the training time of the models roughly consistent across all

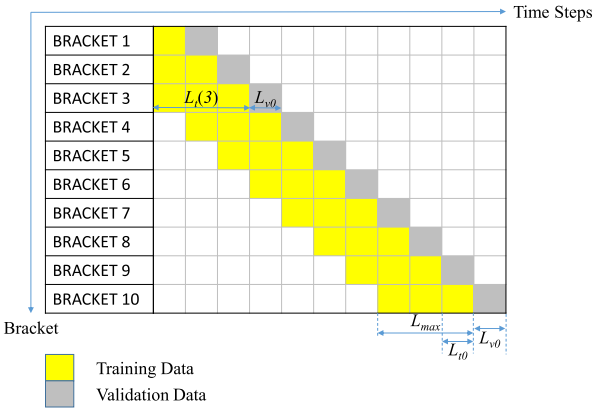

FIGURE 1. Arrangements of D_t and D_V with the progression of successive halving and progressive sampling.

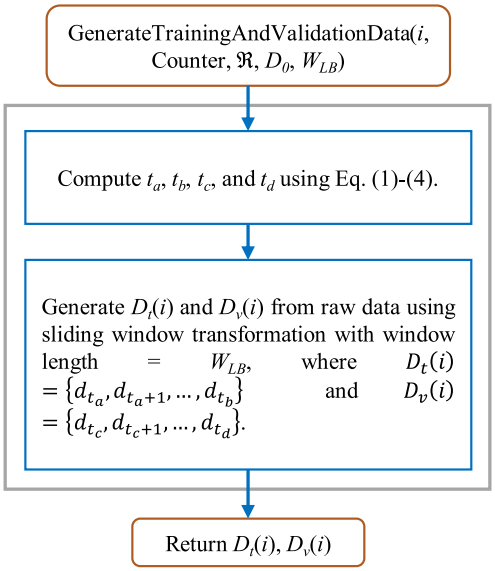

FIGURE 2. Flowchart for generating validation and training dataset.

brackets, the number of epochs, E_p is set to be larger when the size of $D_t(i)$ is small, and smaller when the size of $D_t(i)$ is large. Generally, E_p is computed as follows:

$$E_p = E_{p0} \times \frac{\text{len}\,(\mathcal{M}_1)}{\text{len}\,(\mathcal{M}_i)} \times \frac{L_{\max}}{L_t(i)} \tag{5}$$

where E_{p0} is a positive integer, len$(\cdot)$ is the length of a list, and $\mathcal{M}_i$ is the collection of models in the ith bracket. In this work, the total number of models generated in the first bracket, len $(\mathcal{M}_1)$ is K_0, and the value of E_{p0} is one. The early stopping technique is implemented to prevent overfitting during training. We try to train every model for at least 10 epochs, before applying the early stopping technique. This means that we do not use early stopping when $E_p \leq 10$, and only use early stopping when $E_p > 10$. The patience, P is defined as the number of epochs that we allow the model to be trained for, when the validation loss does not improve, and

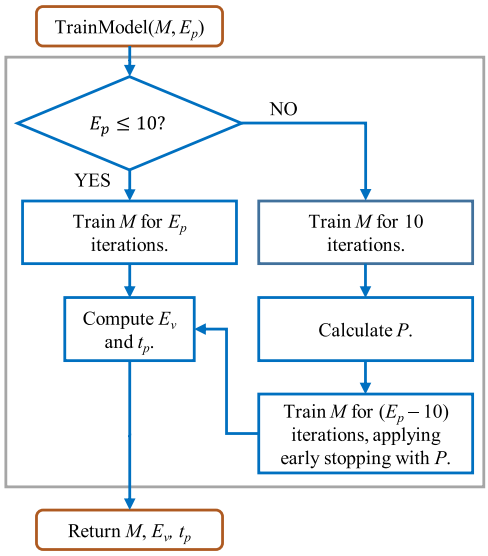

FIGURE 3. Flowchart for training the models.

P is calculated as follows:

$$P = \min\left(\text{ceil}\left(\frac{E_p}{10} - 1\right), 6\right) \tag{6}$$

where min$(\cdot)$ is the smallest number in a list, and ceil$(\cdot)$ means rounding towards positive infinity. With this, we ensure that we do not terminate the training prematurely when E_p is small, while preventing overfitting and saving the budget when E_p is large. The maximum value of P is limited to six to reduce the overall training time. Other than that, the training time per epoch per length of training dataset, t_p of the kth model is recorded as:

$$t_p = \frac{t_m}{E_p \times L_t(i)} \tag{7}$$

where t_m is the total training time of that model, M. Fig. 3 shows the flowchart of the training process named TrainModel(). We use the mean squared error (MSE) as the loss function and the Adam optimizer [38] to update the network weights.

C. BAYESIAN OPTIMIZATION IN THE BRACKETS

In this work, BO is used for two purposes; first to generate models in the first bracket of the proposed ASH-HPO algorithm, and second to add additional models in random brackets so that the surrogate model of the later brackets can contribute by generating some models as well. The surrogate model of the later brackets will be more reliable than the first brackets since it is updated on a larger training dataset. Thus, adding models generated by the surrogate model of the later brackets can help to improve the robustness of the ASH-HPO algorithm by correcting the mistakes made in the first bracket. In this work, we use a Gaussian process regression (GPR) model as the default surrogate model. In the first bracket, we will create and train 10 models where their hyperparameters values are generated by random sampling

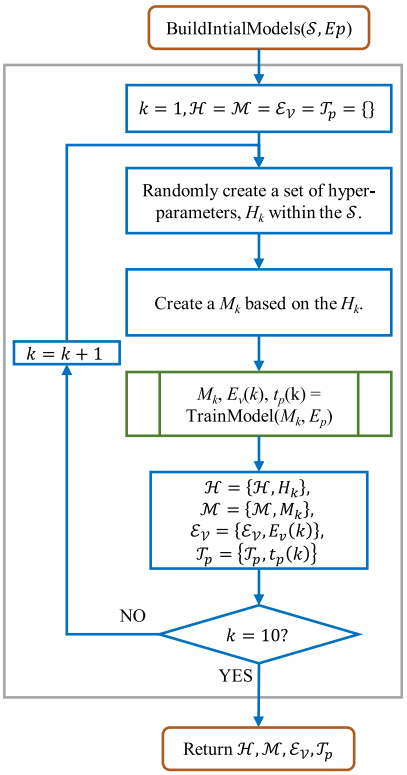

FIGURE 4. Flowchart for building the initial models.

within the hyperparameter space, $\mathcal{S}$ with N hyperparameters. We denote the kth vector of hyperparameters as H_k where $H_1, H_2, \ldots, H_{10} \in \mathcal{H}$, the kth model as M_k where $M_1, M_2, \ldots, M_{10} \in \mathcal{M}$, the validation loss of M_k as $E_v(k)$ where $E_v(1), E_v(2), \ldots, E_v(10) \in \mathcal{E}_v$, and the training time per epoch per length of training dataset as $t_p(k)$ where $t_p(1), t_p(2), \ldots, t_p(10) \in \mathcal{T}_p$.

We will then fit the GPR model, G to the observations made in the first ten models. In other words, we will use M_1 to M_{10} to initialize the GPR model and this model will be used by the BO. The BO process will start by generating an additional model, with its hyperparameters selected by G. Once the training of that model is complete, H, M, $\mathcal{E}_v$ and $\mathcal{T}_p$ are appended with the respective new members, and G is updated with the new observations before it is used to select the hyperparameter values of the next model. This process is repeated until we generate a total of $K_0 - 10$ models through the BO process, which means the total number of models is now K_0 since there are 10 initial models. Fig. 4 shows the flowchart for the function to build the initial models M_1 to M_{10} named BuildInitialModels().

A variable counter is assigned zero initially, and increased by one at the end of each bracket. As described previously, BO will be performed when the counter = $\Re$, before resetting the counter back to zero. For each subsequent time BO is called after the initial models were created, it will generate two less models, starting from 10 and decreasing to one. Let K_m be the number of models generated during the mth time the BO is performed. K_m can be defined as:

$$K_m = \begin{cases} K_0 - 10, & m = 1 \\ 14 - 2m, & 2 \leq m \leq 6 \\ 1, & m > 6. \end{cases} \tag{8}$$

In addition, we double the E_p calculated using (5) during the BO process after the first one ($m > 1$). This is because the models generated during the earlier stage will be trained with more E_p than the newly generated models, thus doubling the E_p prevents a premature termination of training. This also makes the comparison between the old and new models fairer. An open source Python package pyGPGO is used to implement Bayesian Optimization [39].

D. MODIFIED SUCESSIVE HALVING

In this work, we propose a modification to the successive halving algorithm where in each iteration or bracket there are two stages; an odd stage which is a determination stage, and an even stage which is an elimination stage. In the odd stage, we train every surviving model, and record its validation loss. In the even stage, we remove the under performing models by a factor of λ and continue training the surviving models. There is a special case where stage one in bracket one is actually used for BO to generate models. Also, when the counter = $\Re$, the even stage in that bracket is also used to perform BO to generate new models after eliminating $(\lambda - 1)/\lambda$ of the bad performing models so that only $1/\lambda$ of the models survive. Fig. 5 compares the number of surviving models of the modified and original successive halving algorithms across the brackets. It can be seen that the modified successive halving ensures that each model will be trained on the two different training subsets from the current bracket and the previous bracket before being removed, except in the first elimination stage where models are only trained on the dataset of stage 1. This improves the stability of the algorithm at the earlier stages when the training dataset is very small.

Fig. 6 shows an example where L_{max} is $(3 \times L_{t0})$, $K_1 = 16$, $\lambda = 2$, K_2 is 4 during the second BO, $K_3 = 2$ during the third BO, and $\Re$ is initially 4. As can be seen, 16 models are generated in stage one in bracket one through the combination of random search and BO. At the end of the even stage in each bracket, or to be exact after eliminating half of the models, the counter is increased by one. Then, in stage eight of bracket four where the counter = $\Re$, four additional models are generated. After that, the counter is reset to zero again and $\Re$ is given a new random value of two. The counter is equal to $\Re$ once again in stage 12 of bracket 6 and only two new models are generated.

However, it can be a challenge to decide on how to update the surrogate model, G. Since we will remove the worst performing models in every bracket, the validation losses of the models that are removed in the earlier brackets will not be updated anymore. Thus, the E_v of models that are terminated in the earlier brackets no longer accurately represent their

actual validation losses. This is because if the models survive, they will be trained with larger training datasets and with more epochs compared to if they do not. Therefore, it is reasonable to assume that their actual validation losses are much lower than their recorded E_v. In order to solve this problem, we employ a similar method as found in [30] where we use a term named error rate, E_r which is defined as the ratio between the mean of the validation losses of the surviving models in the jth stage, μ_j and the previous stage μ_{j-1} and can be computed as

$$E_r = \begin{cases} 1, & \text{len}\,(\mathcal{M}_i) \leq 4 \\ \dfrac{\mu_j}{\mu_{j-1}}, & \text{otherwise.} \end{cases} \tag{9}$$

Every time a model, M_k is deleted from $\mathcal{M}$, we will delete its validation loss $E_v(k)$ as well from $\mathcal{E}_v$. In order to keep track of the validation losses of the deleted models, we will create another list named $\mathcal{E}'_v$, where $\mathcal{E}'_v = \{E'_v(1), E'_v(2), E'_v(3), \ldots\}$ keeps all the validation losses of deleted and surviving models. At the end of each stage, we will update the validation losses of the removed models as follow:

$$E'_v(k) = E'_v(k) \times E_r, E'_v(k) \in \mathcal{E}'_v \tag{10}$$

where $E'_v(k)$ is the validation loss of the kth model. As shown in (9), we set E_r to be one if the number of surviving models is less than or equal to four because the E_r can be unstable when the number of surviving models is too small.

E. IMPROVING THE BAD MODELS

In this subsection, the process of identifying and retraining the bad models, and removing those that fail to improve are explained. We will only search for the bad models when a special condition, $E_r > 1$ occurs. A value of $E_r < 1$ means that the models are improved on average as compared to the previous stage. Specifically, a value of $E_r > 1$ means that the average validation loss of the current stage is worse than the previous stage. We define a bad model as a model whose validation loss is greater than μ_j. Once a bad model is found, it is retrained with the function TrainModel(). If the model performance improves after the first attempt, it will be updated. Otherwise, we will make two more attempts to improve the model. If the model performance does not improve after the third attempt, the model is removed. This process is completed after we iterate through all the bad models. If at least one model is removed, we will increase L_{max} by L_{t0}. This is because we detect at least one model that cannot be improved by training, which can indicate a deficiency of training data. Thus, we will try to use larger training datasets in the future brackets. The flowchart for improving the bad models is presented in Fig. 7 and is named ImproveBadModels().

F. EXPANDING THE SEARCH SPACE

Traditionally, the search space of a HPO process is fixed throughout the process. In this work, we propose a method to expand the search space to make the selection of initial search space less critical to the final performance of the ASH-HPO algorithm. For an N dimensional search space, $\mathcal{S} = \{S_1, S_2, \ldots, S_i, \ldots, S_N\}$ where S_i represents the search range of the ith hyperparameter, let $S_l(i)$ and $S_u(i)$ be the lower and upper bounds of S_i respectively, where $\{S_l(i), S_u(i)\} = S_i$. If the ith hyperparameter, $H_k(i)$ of the best performing model (we assume that the kth model is the best performing model) is close to $S_l(i)$ or $S_u(i)$, we will expand S_i in the direction of $S_l(i)$ or $S_u(i)$ by 10%. $H_k(i)$ is said to be close to $S_l(i)$ if

$$\mathcal{S}_l(i) \leq H_k(i) \leq \mathcal{S}_l(i) + 0.1 \times [\mathcal{S}_u(i) - \mathcal{S}_l(i)] \tag{11}$$

and close to $\mathcal{S}_u(i)$ if

$$\mathcal{S}_u(i) - 0.1 \times [\mathcal{S}_u(i) - \mathcal{S}_l(i)] \leq H_k(i) \leq \mathcal{S}_u(i). \tag{12}$$

In these cases, we will expand our search space as follows:

$$\mathcal{S}_l(i) = \mathcal{S}_l(i) - 0.1 \times [\mathcal{S}_u(i) - \mathcal{S}_l(i)] \tag{13}$$

if $H_k(i)$ is close to $S_l(i)$ and

$$\mathcal{S}_u(i) = \mathcal{S}_u(i) + 0.1 \times [\mathcal{S}_u(i) - \mathcal{S}_l(i)] \tag{14}$$

if $H_k(i)$ is close to $S_u(i)$. For certain hyperparameters, conditions are also imposed to prevent nonphysical parameters. For example, the learning rate and number of hidden neurons must be greater than zero.

G. PENALTIES

In order to guide our surrogate model, we impose penalties on models with unwanted characteristics. First, we punish models with long training times. Let $t_p(k)$ be the training time per epoch per length of training dataset of the kth model which is calculated as in (7). Usually $t_p(k)$ is in the order of milliseconds. For this penalty, the validation loss of the kth model is updated as follows:

$$E''_v(k) = \frac{E'_v(k)}{\ln(t_p(k))} \tag{15}$$

where $E'_v(k)$ is the validation loss of that model before the penalty, and $E''_v(k)$ is the validation loss of that model after the penalty. In order to enhance the impact of this penalty, E''_v is also used to decide the surviving models in every elimination stage so that the slow training models will have higher chances of being eliminated.

Second, we punish the bad models and their neighbors. As discussed earlier, we will remove $(\lambda - 1)/\lambda$ of the worst performing models at the end of every even stage. Then, we will also select the worst 10 percent of the members of E''_v, and append the original indices of those members to a special list named I_{bad}. The kth model is identified as a neighbor to another model, the mth model, when the hamming distance between them, $D_{ham}(k, m)$ is less than a certain threshold τ ($\tau = 3$ in this work). The hamming distance between two models in $\mathcal{S}$, with N hyperparameters is computed as

$$D_{ham}(k, m) = \sum_{i=1}^{N} \mathfrak{d}_i(k, m) \tag{16}$$

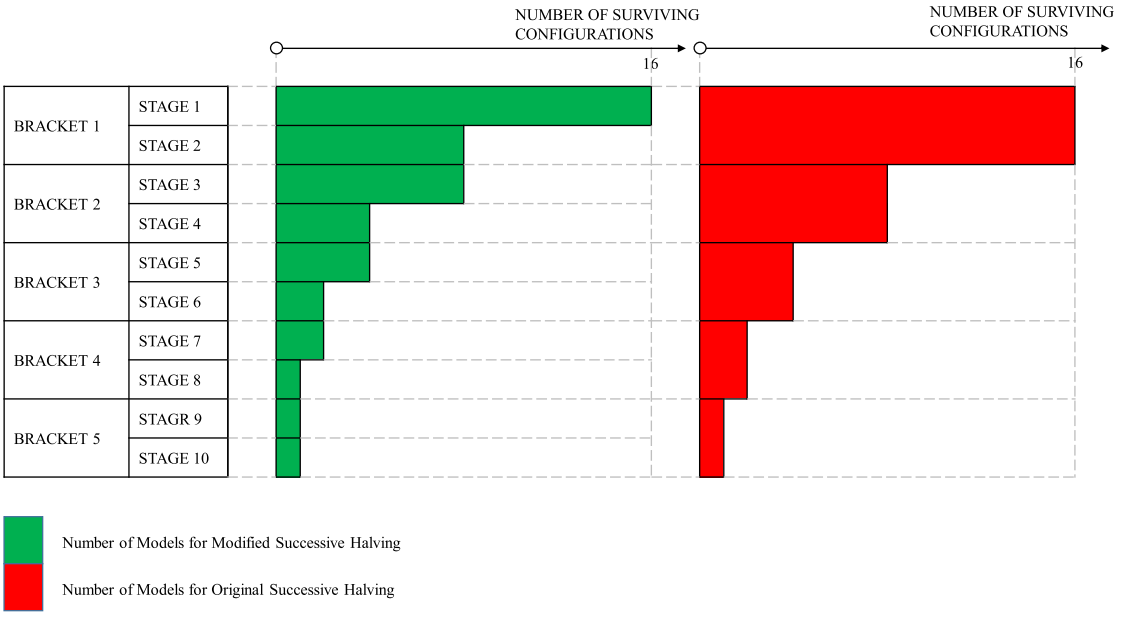

FIGURE 5. Number of surviving models of the modified and original successive halving processes across different brackets.

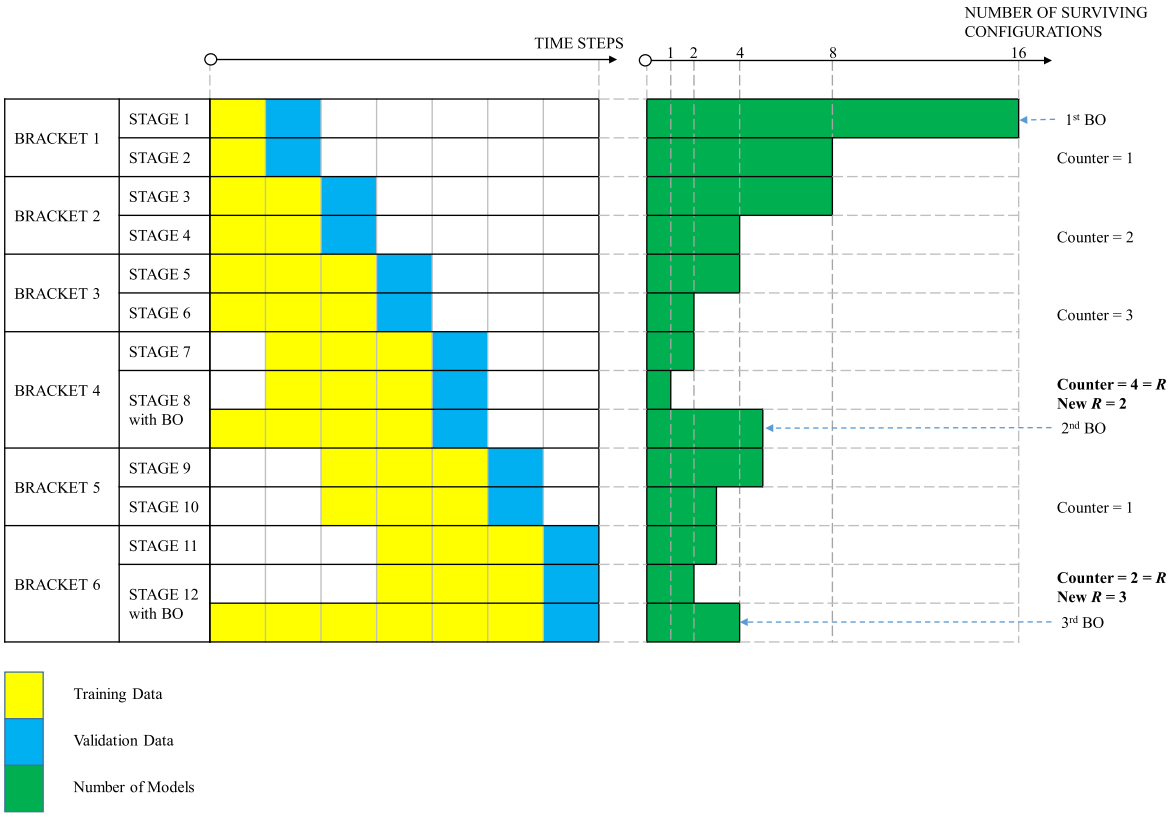

FIGURE 6. Visualization of the change in data arrangement and number of models as the successive halving algorithm progresses.

$$\mathfrak{d}_i(k,m) = \begin{cases} 1, & \dfrac{|H_k(i) - H_m(i)|}{\mathcal{S}_u(i) - \mathcal{S}_l(i)} \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

where $H_k(i)$ and $H_m(i)$ is the ith hyperparameter of the kth and mth model. If the kth model is either a bad model, or it is a neighbor to at least one bad model, we will multiply

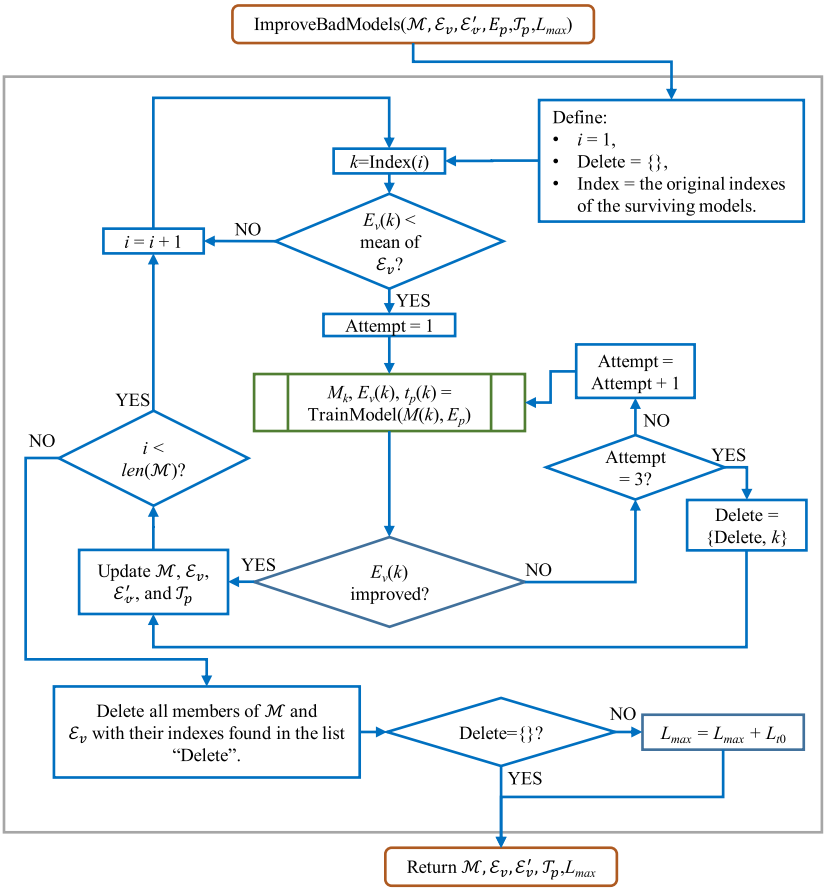

FIGURE 7. Flowchart for improving/retraining the bad models.

its validation loss after the first penalty, $E''_v(k)$ by a penalty factor, F_p ($F_p = 1.2$ in this work). Otherwise we keep its validation loss the same. We compute the validation loss after the second penalty, $E'''_v(k) \in \mathcal{E}'''_v$ as follows:

$$E'''_v(k) = F_p \times E''_v(k). \tag{18}$$

Basically, both penalties modify the optimization problem from optimizing E'_v which is the validation loss, to optimizing E'''_v which is a function of the validation loss, training time, and how close the hyperparameters are to the bad configurations. The first penalty is used to guide the G away from slower training configurations, and the second penalty is used to guide the G away from bad performing configurations and other configurations that are very close to them. Table 1 shows how the values of $E'_v(k)$ and $t_p(k)$ can affect the value of $E''_v(k)$, which is directly related to the objective function to be optimized. It can be seen that the model with lower $E'_v(k)$ and lower $t_p(k)$ has higher $E''_v(k)$. Fig. 8 shows the flowchart for implementing BO, named function BayesianOptimization().

H. OVERALL WORKFLOW

The remaining functions are summarized here. Fig. 9 shows a flowchart for training all surviving models named function TrainAllModels(). This function is used to call function TrainModels() repeatedly to train every models in $\mathcal{M}$. Fig. 10 shows a flowchart for initialization of the overall workflow named function Initialize(). This function is used to define every input to the ASH-HPO algorithm. Finally, the flowchart for the whole process is shown in Fig. 11. The termination criteria of the ASH-HPO algorithm is based on two conditions: when the goal is met (terminated and converged) or when the budget is used up (terminated but not converged). Each time the validation loss is updated, the ASH-HPO algorithm will compare it to the goal and terminate itself if the validation loss is lower than the goal. The budget is limited to prevent the algorithm from running indefinitely.

TABLE 1. Comparisons between different values of $E_v(k)$ and $t_p(k)$.

Example	$E'_v(k)$	$t_p(k)$	$E''_v(k)$
1st Example	1	e^{-10}	-0.1
	0.1	e^{-10}	-0.01
2nd Example	1	e^{-10}	-0.1
	1	e^{-100}	-0.01

IV. RNN, LSTM, CNN, AND CNN-LSTM

A RNN is a type of a neural network that has an internal memory. Unlike a feedforward network, the RNN has a

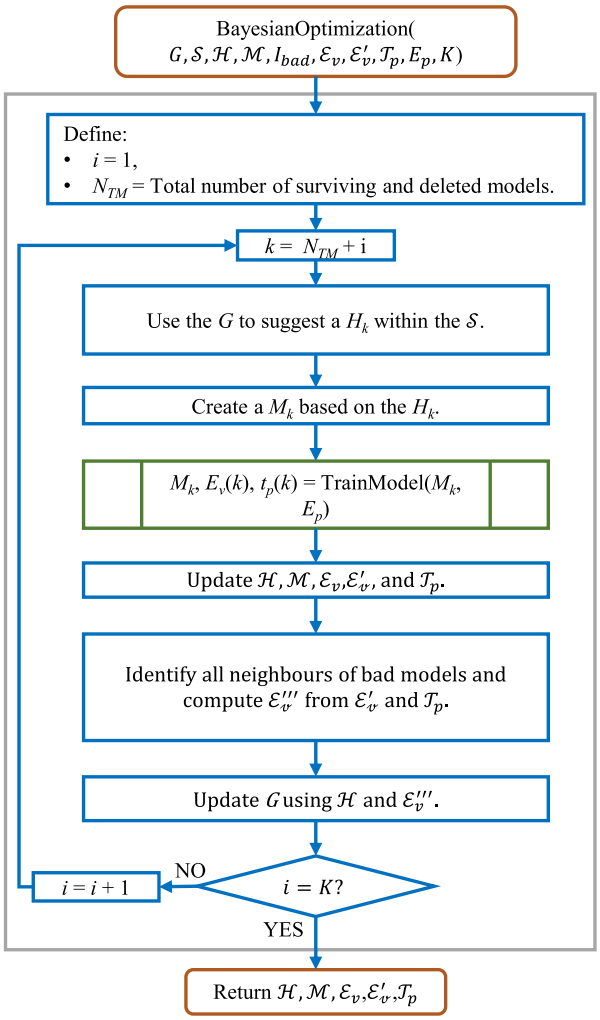

FIGURE 8. Flowchart for implementing BO.

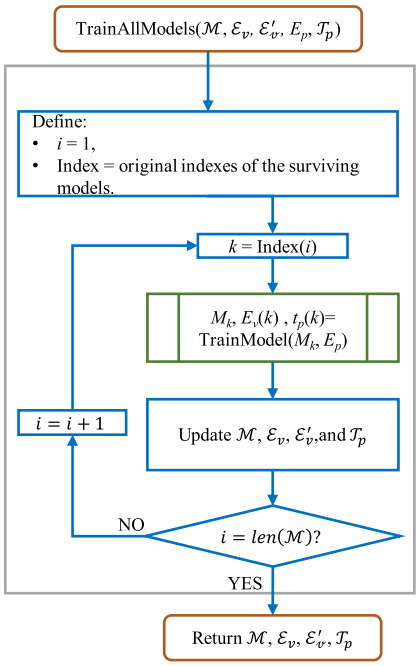

FIGURE 9. Flowchart for training every model in $\mathcal{M}$.

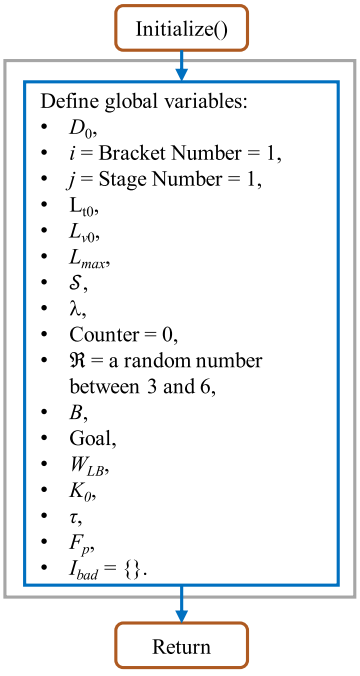

FIGURE 10. Flowchart for initializing the ASH-HPO algorithm.

feedback loop connected to its past decisions, which means that its outputs from the previous time steps will be used as inputs for the current time step. This is what allows it to have memory, and this memory is what makes it a great candidate for dealing with time series data. RNNs can be very challenging to train due to vanishing or exploding gradient problem [24], [40]. This is because the gradients in deeper layers are calculated as products of differentials. Trained over a long time, this can lead to an exploding or vanishing gradient, depending on whether the individual gradients are more than or less than one. Thus, RNNs may be unsuitable for long sequences and long-term dependencies.

The LSTM neural network is a type of RNN which was developed to solve the training problems in traditional RNNs. An LSTM cell has 3 gates: a forget gate, an input gate, and an output gate. The forget gate, f_t decides which information needs attention and which can be ignored. The input gate, i_t decides what information is relevant to update in the current cell state, c_t. The output gate, o_t determines the value of the next hidden state, h_t. An LSTM network can be controlled to remember or forget the dependency on individual inputs by changing the values of i_t, f_t, and o_t. Further background on RNN and LSTM can be found in [40]–[42].

The CNN-LSTM is a hybrid network created by combining a CNN and an LSTM network [31]. The CNN-LSTM network consists of one or more one-dimensional convolution layer, followed by a max pooling layer, and the output is then flattened to feed into one or more LSTM layers. Finally, the outputs of these layers are fed into one or more

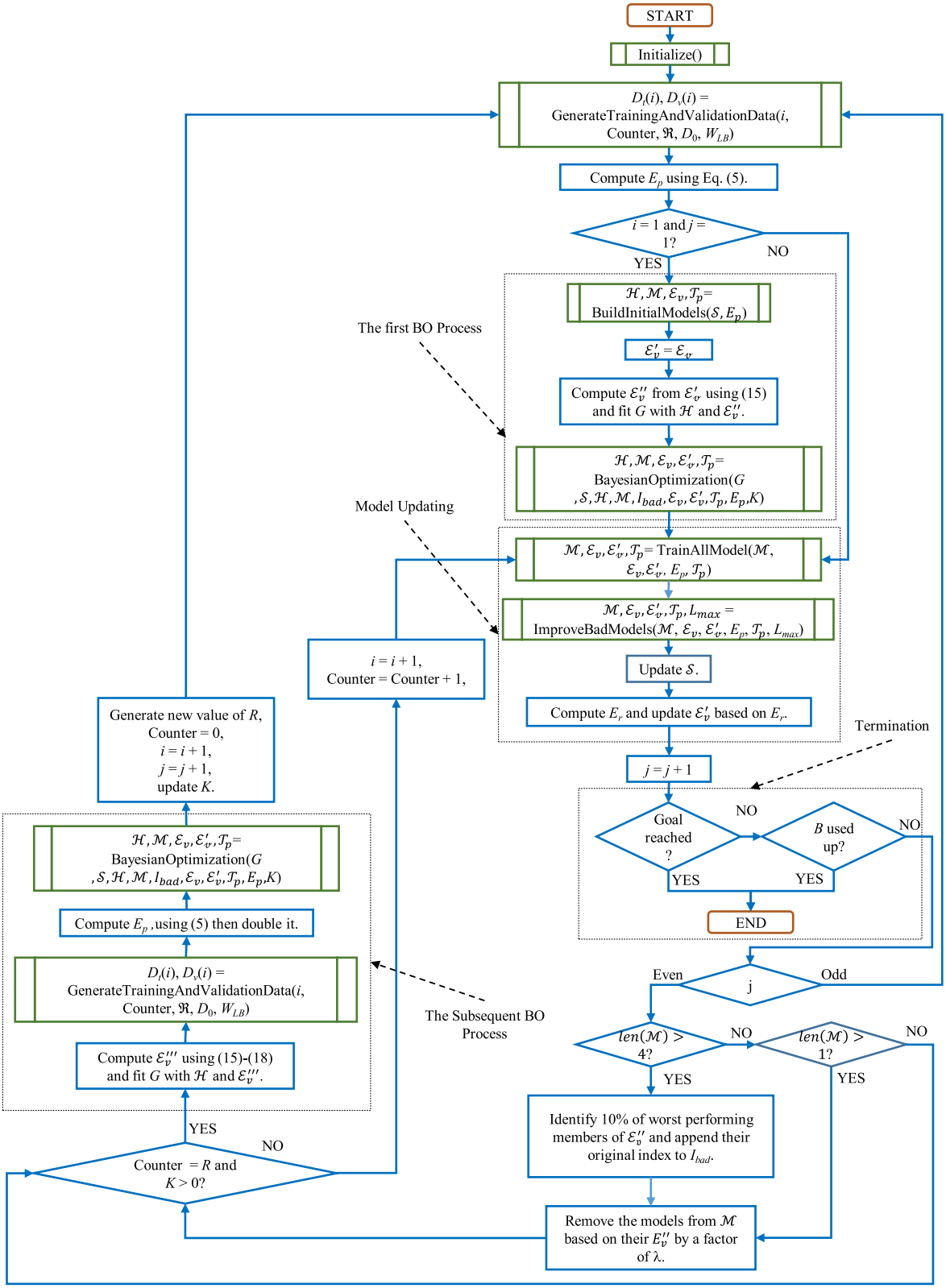

FIGURE 11. Overall workflow for the ASH-HPO algorithm.

fully-connected/dense layers. The last dense layer is the output layer, and its nodes have linear activation functions. Fig. 12 shows the structure of a CNN-LSTM model. The figure also shows the visual representation of an LSTM cell,

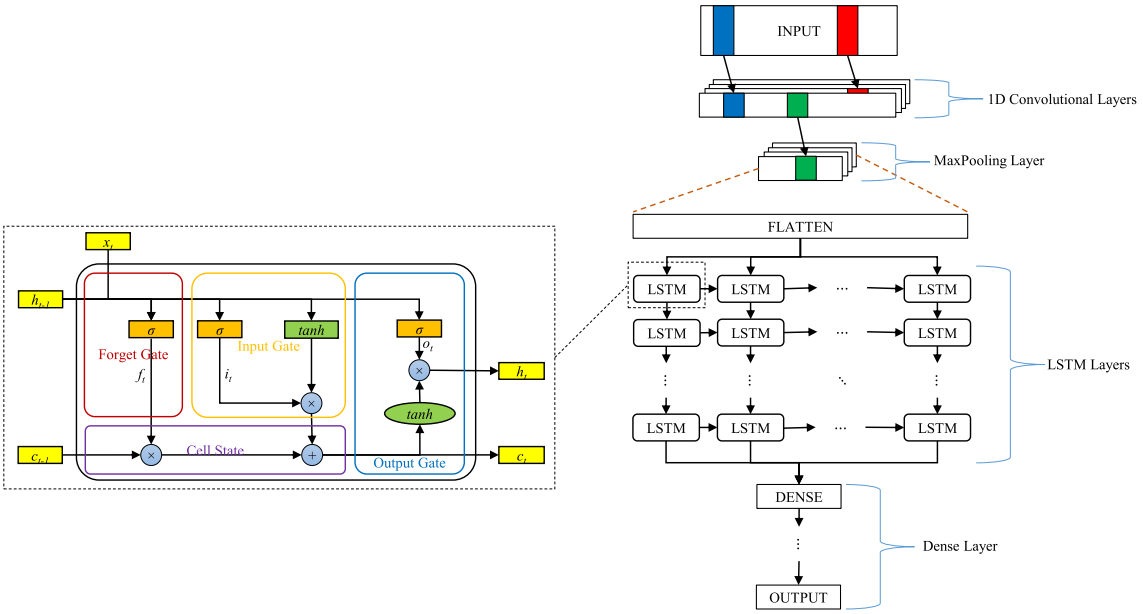

FIGURE 12. The structure of a CNN-LSTM model and an LSTM cell.

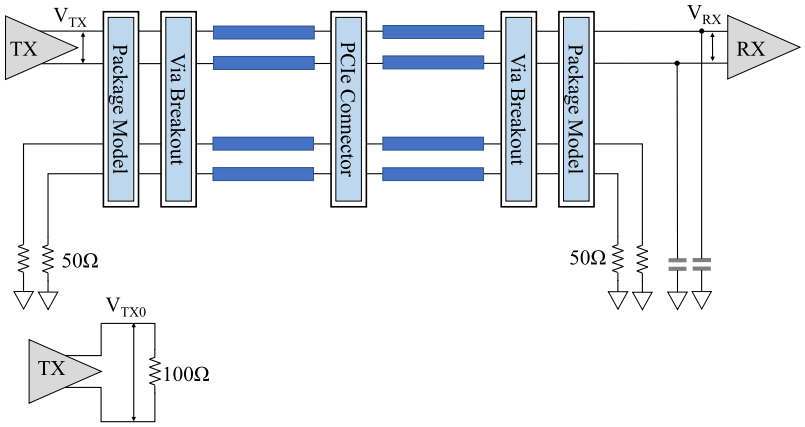

FIGURE 13. The PCIe Gen 2 topology.

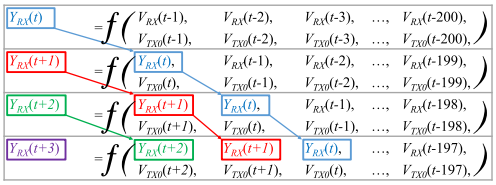

FIGURE 14. Rolling forecast method.

which is the building block of the LSTM layers. Compared to the CNN-LSTM network, a CNN model can be constructed in a similar method by connecting its flatten layer directly to the first dense layer since it has no LSTM layers.

V. NUMERICAL EXAMPLES

In this section, we present three examples to illustrate the strengths of the proposed ASH-HPO algorithm. The first example uses a PCIe Gen 2 channel, the second example uses a PCIe Gen 5 channel, and the third example uses a PAM4 differential channel. The effects of the different settings of the algorithm such as the penalties and parameters of the algorithm are also investigated. As a benchmark, the ASH-HPO algorithm is also compared to the BO, successive halving, and hyperband methods. All experiments in this paper are performed using a computer with an Intel® Core™ i7-10700k Processor @3.8GHz, 32 GB RAM, and NVIDIA GeForce RTX 2080 SUPER.

A. PCIe GEN 2 CHANNEL

The PCIe Gen 2 topology is shown in Fig. 13. V_{TX} is the output voltage of the transmitter (TX), V_{RX} is the input voltage to the receiver (RX), and V_{TX0} is the output voltage of TX when it is terminated with a 100 Ω resistor. The TX outputs a PRBS bit sequence with a rate of 5 Gbps with

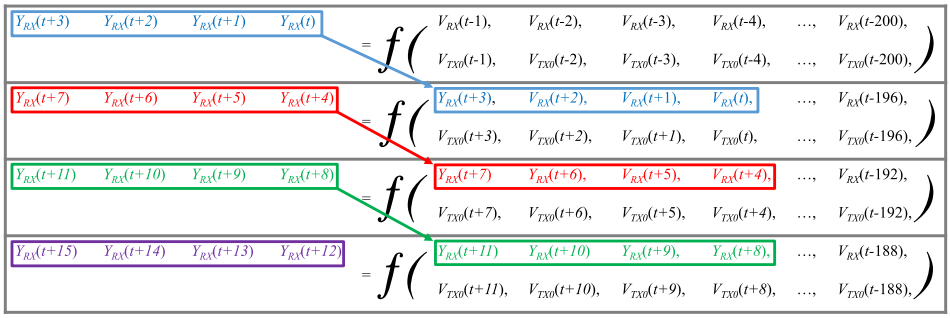

FIGURE 15. Rolling forecast method with $W_{LA} = 4$.

an 8b/10b encoding. The voltage waveforms are sampled with a sampling rate of 10 samples per bit during a transient simulation. The dataset for this example is generated from the voltage waveforms of a million bits, where we use the earlier bits for training and validation, and the remaining bits for testing. We define our problem as using V_{TX0} and V_{RX} from a time step of $(t-1)$ to $(t-W_{LB})$ to predict $Y_{RX}(t) \approx V_{RX}(t)$ where the look back window, $W_{LB} = 200$. Using a larger value of W_{LB} will give a better performance, but it requires more time for data preprocessing and training, and we find 200 to be a suitable number as it gives a good performance while not being computationally expensive.

During the prediction process, we will use the rolling forecast method, which involves feeding the current prediction back into the window to make a prediction at the next time step. Fig. 14 shows the formulation of this problem and demonstrates the rolling forecast method that predicts a single time step into the future. This method can also be applied for multi-step forecasting problems by defining a look ahead window, W_{LA} as the number of time steps into the future during the rolling forecasting process. For example, Fig. 15 demonstrates the rolling forecast prediction process for $W_{LA} = 4$. The single time step rolling forecast method is basically the multi-step rolling forecast method with $W_{LA} = 1$. The $\mathcal{S}$ is an 12-dimensional search space which includes the learning rate, number of convolution layers, number of LSTM layers, number of LSTM nodes, dropout rate, filter size, kernel height, pool size, choices of activation functions where the choices are the rectified linear unit (ReLU), hyperbolic tangent (tanh), exponential linear unit (ELU), and scaled exponential linear unit (SELU), number of dense layers, number of dense nodes, and batch size. V_{TX0} and V_{RX} are normalized before being fed to the neural networks. The goal is: validation loss $\leq$ 1e-4. All the experiments are repeated 10 times to obtain better estimations of the results.

We investigate the effects of different algorithm parameters and settings on the performance of the ASH-HPO algorithm. First, we investigate the effects of penalties by using four cases: no penalty, punish slow models only, punish bad models only, and both penalties. We set the number of initial models, $K_0 = 81$, elimination factor, $\lambda = 3$, $L_{t0} = L_{v0} = 2$k,

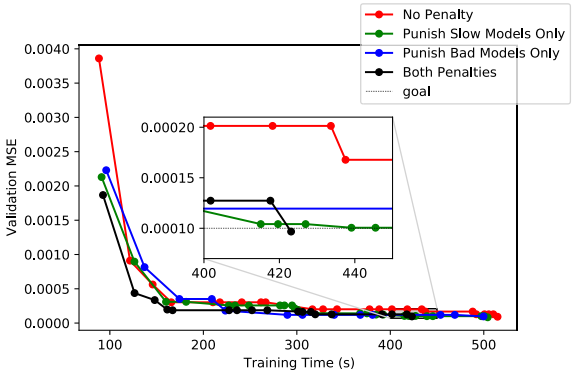

FIGURE 16. Validation losses of the ASH-HPO algorithm with different penalty settings versus training time.

TABLE 2. The effect of penalties on the ASH-HPO algorithm.

Punish Slow Model	**No**	**No**	**Yes**	**Yes**
Punish Bad Model	**No**	**Yes**	**No**	**Yes**
Total time steps of training data	48k	50k	26k	38k
Testing loss	9.59e-5	9.52e-5	9.44e-5	9.18e-5

TABLE 3. The effect of W_{LA} on the ASH-HPO algorithm.

Value of W_{LA}	**1**	**20**	**50**	**100**
Total training time (s)	368	467	799	1100
Testing loss	1.02e-4	9.18e-5	1.12e-4	1.11e-4
Prediction time per time step (s)	8.01e-4	4.20e-5	1.78e-5	1.05e-5
Total time steps of training data	28.2k	39.8k	70k	128.8k

acquisition function = EI, and initial $L_{max} = (3 \times L_{t0})$. Table 2 shows the results and the convergence rate plots are visualized in Fig. 16, where it can be seen that the case "both penalties" converges the fastest, and the case "no penalty" converges the slowest. It can be seen from the table that the first penalty can reduce the training time despite using more training data as the training time per time step is shorter, and the second penalty can reduce the amount of training data required to reach the goal. Combining both penalties gave the best results. Thus, we will apply both penalties for the ASH-HPO for the rest of this paper.

Then, we investigate the effects of the W_{LA} value ($W_{LA} = 1, 20, 50, 100$) on the ASH-HPO algorithm. We maintain all

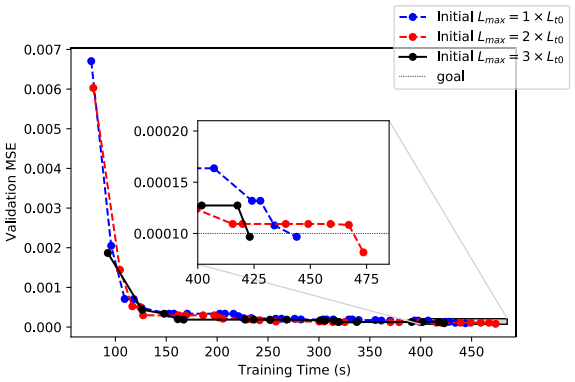

FIGURE 17. Validation losses of the ASH-HPO algorithm with different initial L_{max} versus training time.

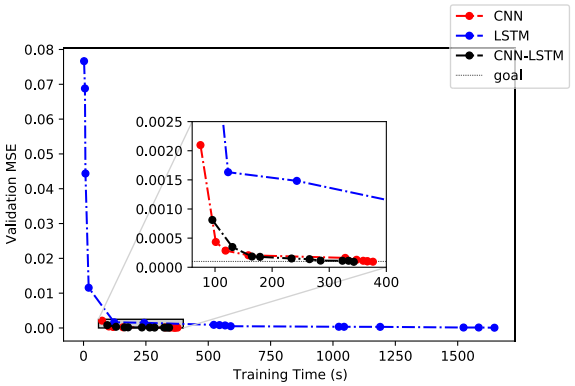

FIGURE 18. Validation losses of the ASH-HPO algorithm using CNN, LSTM and CNN-LSTM models versus training time.

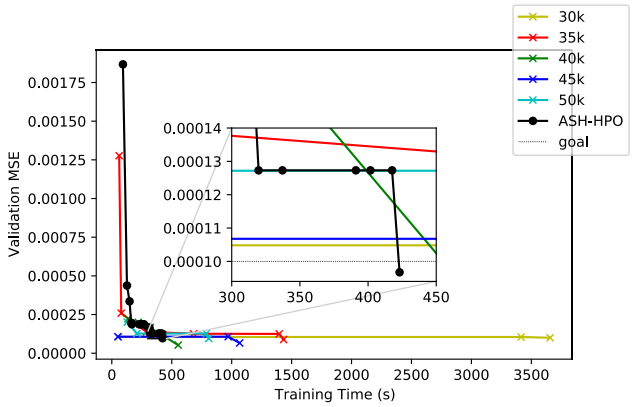

FIGURE 19. Validation losses of the ASH-HPO algorithm and BO algorithm with different L_t versus training time.

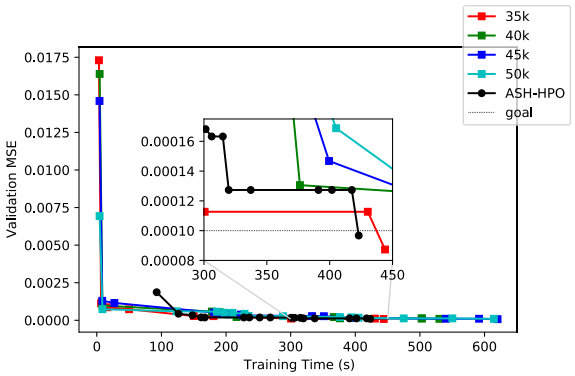

FIGURE 20. Validation losses of the ASH-HPO algorithm and successive halving algorithm with different L_t versus training time.

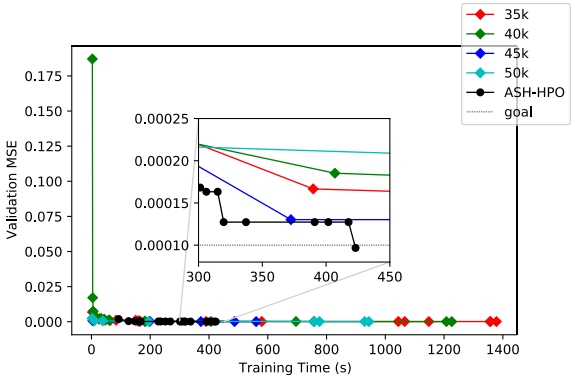

FIGURE 21. Validation losses of the ASH-HPO algorithm and hyperband algorithm with different L_t versus training time.

TABLE 4. The effect of acquisition functions on the ASH-HPO algorithm.

Acquistition Function	EI	Entropy	UCB	PI
Total training time (s)	423	576	481	498
Testing loss	9.18e-5	9.56e-5	9.26e-5	1.02e-4
Total time steps of training data	39.8k	54.2k	46.8k	45.5k

other algorithm parameters from the case "both penalties" and only change the value of W_{LA}. The results are tabulated in Table 3. As the value of W_{LA} increases, the ASH-HPO algorithm requires more time to converge due to the increase in complexity of the models. Other than that, increasing the value of W_{LA} also increases the total time steps of training data required to reach the goal, which can slow down the convergence rate even more. However, using larger W_{LA} values can greatly improve the prediction time per time step because each prediction can be used to predict more time steps ahead, thus requiring fewer predictions to predict the whole sequence. For the PCIe Gen 2 example, we use $W_{LA} = 20$ because it gives the best trade-off between the training and prediction processes.

Next, we investigate the effects of the initial L_{max} value, using 3 cases: $L_{max} = (1 \times L_{t0})$, $L_{max} = (2 \times L_{t0})$, and $L_{max} = (3 \times L_{t0})$. In these cases, we maintain all other algorithm parameters. L_{max} is a variable that decides the maximum length of the training subset in each bracket, and its value is increased when the function ImproveBadModels() detects any bad model that cannot be improved. The results are visualized in Fig. 17, which shows that the case $L_{max} = (3 \times L_{t0})$ is slightly faster than the rest of the pack in terms of the convergence rate, and it requires the least number of training data, using only 38k time steps as compared to 79k time steps for the case $L_{max} = (1 \times L_{t0})$, and 62k time steps for the case $L_{max} = (2 \times L_{t0})$. The results also show that the convergence rate of the ASH-HPO algorithm is only

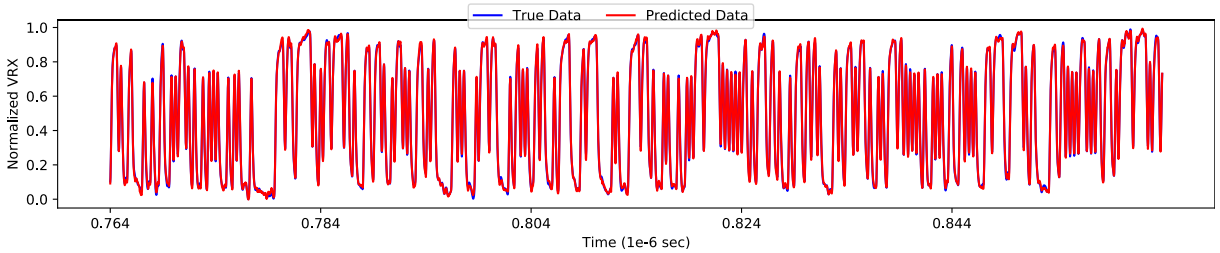

FIGURE 22. True and predicted normalized V_{RX} waveforms of the PCIe Gen 2 channel for time step, $t = 1$ to $t = 5$k of the testing dataset, testing loss = 1.04e-4.

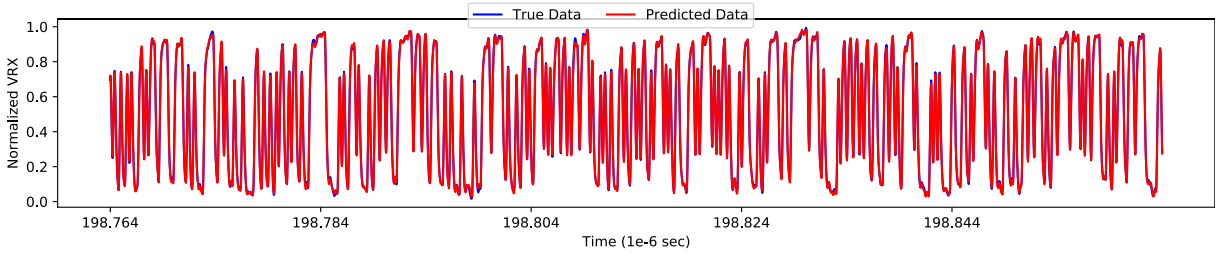

FIGURE 23. True and predicted normalized V_{RX} waveforms of the PCIe Gen 2 channel for time step, $t = 9900\text{k} + 1$ to $t = 9905$k of the testing dataset, testing loss = 1.03e-4.

TABLE 5. The hyperparameters search space and the best configurations for the CNN, LSTM, and CNN-LSTM models for the PCIe Gen 2 example.

Hyperparameters	CNN			LSTM			CNN-LSTM		
	Min	Max	Best	Min	Max	Best	Min	Max	Best
Learning rate	10^{-5}	10^{-2}	4.97×10^{-5}	10^{-5}	10^{-2}	9.15×10^{-4}	10^{-5}	10^{-2}	1.86×10^{-4}
No. of conv. layers	1	2	1	—	—	—	1	2	1
Filter size	8	64	60	—	—	—	8	64	53
Kernel height	8	64	37	—	—	—	8	64	34
Pool size	1	8	2	—	—	—	1	8	2
No. of LSTM layers	—	—	—	1	2	1	1	2	1
No. of LSTM nodes	—	—	—	16	128	[116]	16	128	[113]
Dropout rate	0	0.3	0.1425	0	0.3	0.1969	0	0.3	0.0977
No. of dense layers	0	2	1	0	2	1	0	2	0
No. of dense nodes	16	128	[77]	16	128	[103]	16	128	—
Batch size	2^5	2^{10}	2^7	2^5	2^{10}	2^5	2^5	2^{10}	2^8

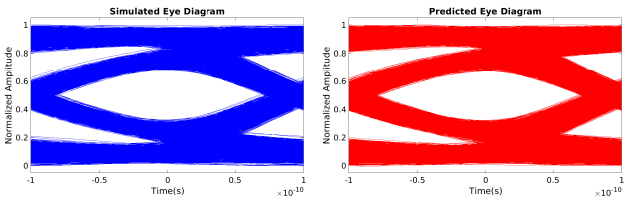

FIGURE 24. Eye diagrams generated using the (a) simulated and (b) predicted voltage waveform for the PCIe Gen 2 channel.

marginally affected by the choice of initial L_{max} as all three cases have very similar training times. A case with smaller initial L_{max} trains the models with a smaller data subset, but requires more brackets to reach the goal, which increases the size of data required for the convergence.

Then, we compare the performance of the ASH-HPO algorithm when it is used with different types of neural networks. In this work, we investigated the CNN, LSTM, and CNN-LSTM models. Fig. 18 compares the convergence rates of the cases using the CNN, LSTM, and CNN-LSTM networks. We use $W_{LA} = 1$ for all the cases since higher values require very high computational time and memory for the LSTM case. We remove the hyperparameters of the LSTM layers from $\mathcal{S}$ for the CNN models and we remove the hyperparameters of the convolutional layers from $\mathcal{S}$ for the LSTM models. We maintain the other algorithm parameters. It can be seen that the ASH-HPO algorithm with CNN-LSTM converges much faster than the ASH-HPO algorithm with LSTM. This is because the training time per epoch for the CNN-LSTM models is much shorter than that of the LSTM models. The convergence rate for CNN-LSTM is also slightly faster than that of the CNN. The CNN-LSTM models require only 8.01e-4 second on average to perform prediction for each time step. On the other hand, the LSTM models require 1.53e-2 second to predict one time step which is approximately 19 times slower than the CNN-LSTM models. The prediction time of CNN is about 6.34e-4 second per time step, which is slightly faster than the CNN-LSTM model. However, the testing loss of the CNN-LSTM model is 1.03e-4, which is lower than the CNN, which is 1.16e-4.

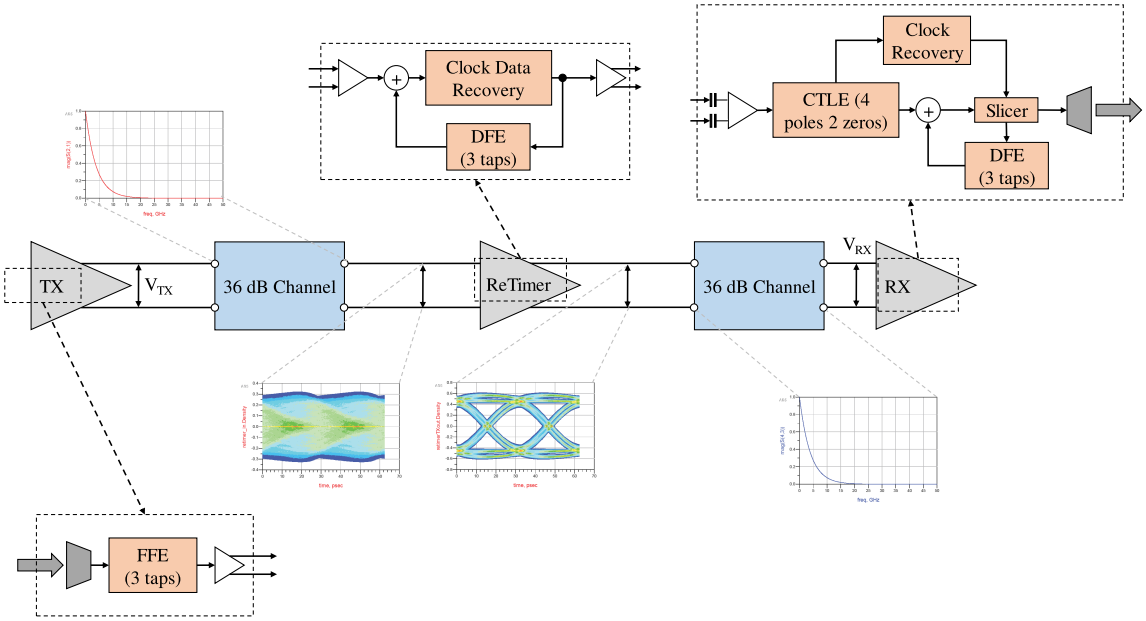

FIGURE 25. The PCIe Gen 5 topology.

Next, we investigate the effects of different acquisition functions which are EI, entropy search, UCB, and PI on the performance of the ASH-HPO algorithm. The results are tabulated in Table 4. It can be seen that the ASH-HPO algorithm using EI outperforms the other acquisition functions, as it converges the fastest, requires the smallest size of training data to reach the goal, and has the lowest testing loss.

Besides that, we also compare the proposed ASH-HPO algorithm to other existing methods. Firstly, we compare the ASH-HPO algorithm to the BO algorithm without the progressive sampling strategy. For the BO method, we use 10 random configurations to initialize the algorithm, and we terminate the BO algorithm when the total number of models exceeds 100. Each model is trained up to $E_p = 100$ with $P = 6$. We test the BO algorithm using 10 cases, each case with a different training dataset length, $L_t = 5k, 10k, 15k, \ldots, 50k$ and a fixed validation dataset length, L_v of 2k. The progressive sampling method is not applied to any of them. The convergence plots are shown in Fig. 19. We only show $L_t \geq 30k$ in the figure because all the cases of $L_t < 30k$ fail to reach the goal. The graph shows that the ASH-HPO algorithm converges faster than all of the BO cases.

Next, we compare the ASH-HPO algorithm with the successive halving algorithm. We set the successive halving algorithm to start with 81 initial models, with $\lambda = 3$, $E_{p0} = 1$, and P calculated as in (6). The successive halving algorithm is tested using 10 cases, each with a different $L_t = 5k, 10k, 15k, \ldots, 50k$ value and a L_v value of 2k. The convergence plots are shown in Fig. 20. We only show $L_t \geq 35k$ in the figure because all the cases of $L_t < 35k$ fail to reach the goal. The graph shows that the ASH-HPO algorithm converges faster than all of the successive halving cases.

Finally, we compare the ASH-HPO algorithm with the hyperband algorithm. For the hyperband algorithm, we set the inputs of the hyperband algorithm to be $K_0 = 81$, $\lambda = 3$, and P is calculated using (6). The hyperband algorithm is tested using 10 cases, each with a different $L_t = 5k, 10k, 15k, \ldots, 50k$ value and a L_v value of 2k. The convergence plots of cases that reach the goal are shown in Fig. 21. The graph shows that the ASH-HPO algorithm has a faster convergence speed than all of the hyperband cases.

In addition to faster convergence speed shown in these examples, a main advantage of the ASH-HPO algorithm over the other three algorithms is that it does not require the user to input the value of L_t, since it uses a progressive sampling strategy. This is a significant advantage as different L_t values can have a large impact on the convergence speed and testing loss as can be seen from the examples, and the optimal L_t value cannot be known without performing the actual training. In other words, the ASH-HPO algorithm is able to perform a fast and accurate automated training of the neural models, while limiting trial and error.

Since the ASH-HPO algorithm uses 38k training samples for training and validation, we will use the remaining samples for testing. We use the CNN-LSTM model generated by the

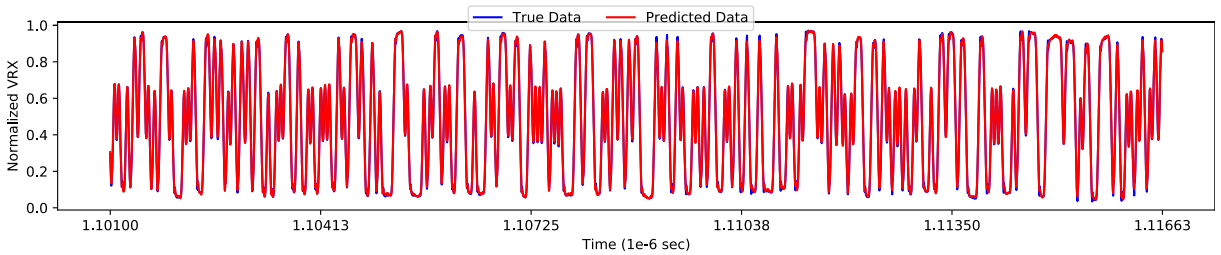

FIGURE 26. True and predicted normalized V_{RX} waveform of the PCIe Gen 5 channel for time step, $t = 1$ to $t = 5$k of the testing dataset, testing loss = 6.97e-5.

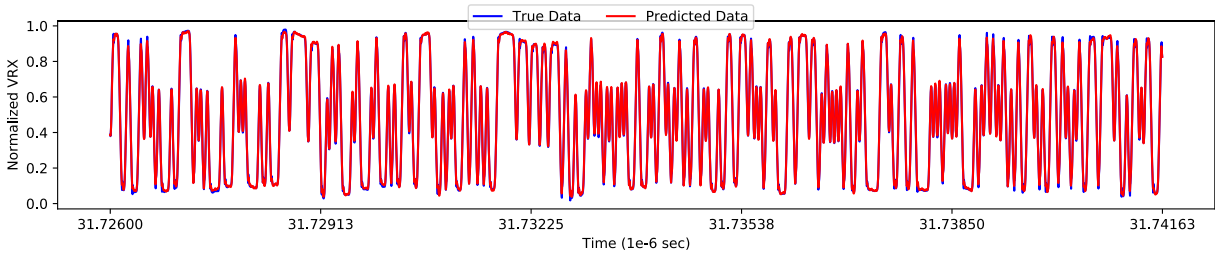

FIGURE 27. True and predicted normalized V_{RX} waveform of the PCIe Gen 5 channel for time step, $t = 9800\text{k} + 1$ to $t = 9805$k of the testing dataset, testing loss = 7.13e-5.

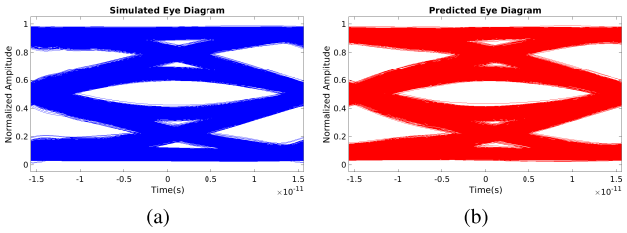

FIGURE 28. Eye diagram generated using the (a) simulated and (b) predicted voltage waveform for the PCIe Gen 5 channel.

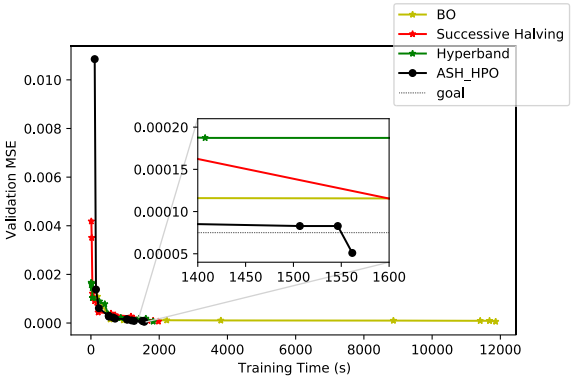

FIGURE 29. Validation losses of multiple HPO algorithms versus time for the PCIe Gen 5 channel.

ASH-HPO algorithm with both penalties, initial $L_{max} = (3 \times L_{t0})$, acquisition function = EI, $W_{LB} = 200$, and $W_{LA} = 20$. Fig. 22 and Fig. 23 show the comparisons between the actual and predicted normalized V_{RX} of the PCIe Gen 2 channel for time step, $t = 1$ to $t = 5$k and $t = 9900\text{k} + 1$ to $t = 9905$k respectively. No significant accuracy degradation is observed from the first testing data subset to the last. The prediction speed is about 42 microseconds per time step. Fig. 24 compares the eye diagrams constructed

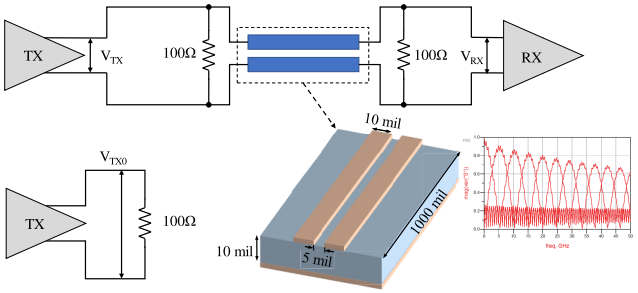

FIGURE 30. The PAM4 differential channel topology.

using the voltage waveforms generated from a transient circuit simulator and from the CNN-LSTM network, and both of them are almost identical. The normalized eye heights for the simulated and predicted eye diagrams are both 0.343, while the eye widths for the simulated and predicted eye diagrams are 14.69×10^{-11} second and 14.60×10^{-11} second respectively. The hyperparameters search space and the best configurations for the CNN, LSTM, and CNN-LSTM networks for the PCIe Gen2 example are tabulated in Table 5.

B. PCIe GEN 5 CHANNEL

The PCIe Gen 5 topology used in this example is shown in Fig. 25. The TX outputs a PRBS bit sequence with a rate of 32 Gbps. The TX uses a feed-forward equalizer (FFE) whereas the RX uses a decision feedback equalizer (DFE) and a continuous time linear equalizer (CTLE). There are two transmission channels, both with 36 dB losses, and a retimer connects the two. V_{TX} is the output voltage of the TX, V_{RX} is the output voltage of the RX, and V_{TX0} is the output voltage of TX when it is terminated with a 100 Ω

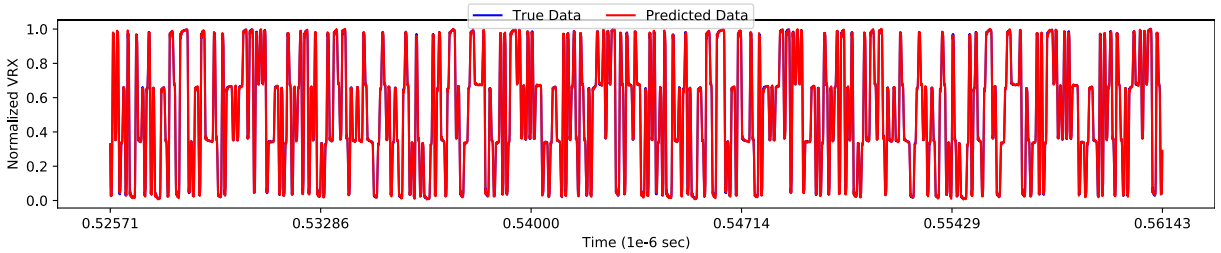

FIGURE 31. True and predicted normalized V_{RX} waveform of the PAM4 differential channel for time step, $t = 1$ to $t = 5$k of the testing dataset, testing loss = 1.65e-5.

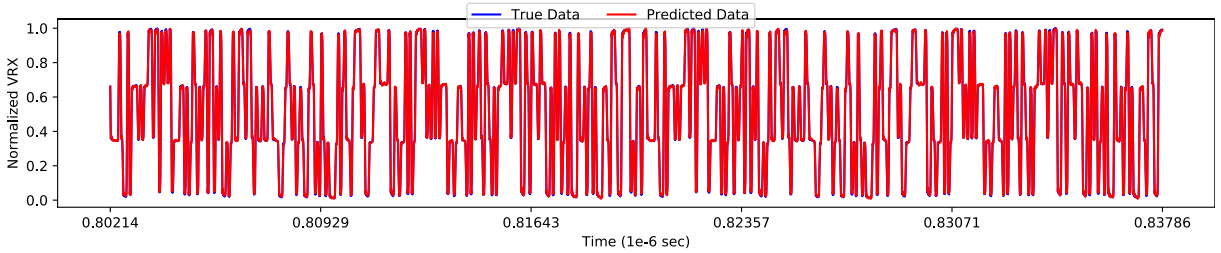

FIGURE 32. True and predicted normalized V_{RX} waveform of the PAM4 differential channel for time step, $t = 400\text{k} + 1$ to $t = 405$k of the testing dataset, testing loss = 1.67e-5.

resistor. The voltage waveforms are sampled with a sampling rate of 20 samples per bit during a transient simulation. The modeling problem is defined as using V_{TX0} and V_{RX} from time steps $(t - 1)$ to $(t - W_{LB})$ to predict the values of V_{RX} from time steps t to $(t + W_{LA} - 1)$, where the look back window, $W_{LB} = 800$, and the look ahead window, $W_{LA} = 10$. The CNN-LSTM model is generated by the ASH-HPO algorithm with the settings: apply "both penalties", number of initial models = 81, $\lambda = 3$, initial $L_{max} = (3 \times L_{t0})$, acquisition function = EI, and a termination goal of validation loss $\leq$ 7.5e-5. A training dataset with 116k time steps is used with a training time of 1561 seconds and a validation loss of 5.10e-5. Fig. 26 and Fig. 27 show the comparisons between the actual and predicted normalized V_{RX} of the PCIe Gen 5 channel for time step, $t = 1$ to $t = 5$k and $t = 9800\text{k} + 1$ to $t = 9805$k of the testing dataset respectively. The prediction speed is about 110 microseconds per time step. No significant accuracy degradation is observed from the first to the last testing set. Fig. 28 compares the eye diagrams generated using the simulated and predicted voltage waveforms. The normalized eye heights for the simulated and predicted eye diagrams are 0.176 and 0.160 respectively, while the eye widths for the simulated and predicted eye diagrams are 2.28×10^{-11} second and 2.21×10^{-11} second respectively. Fig. 29 shows a comparison between the ASH-HPO algorithm and other HPO algorithms which were trained using the same 116k time steps, where it can be seen that the ASH-HPO algorithm converges the fastest.

C. PAM4 DIFFERENTIAL CHANNEL

The PAM4 channel topology used in this example is shown in Fig. 30. The transmission channel is a differential microstrip line with a line width = 10 mil, line length = 1000 mil, separation between two lines = 5 mil,

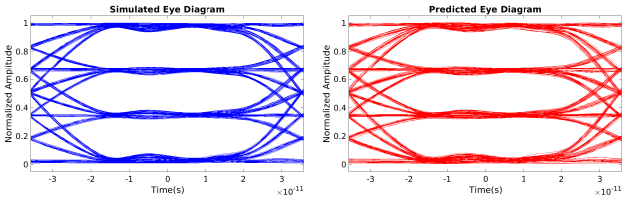

FIGURE 33. Eye diagram generated using the (a) simulated and (b) predicted voltage waveform for the PAM4 differential channel.

TABLE 6. Eye diagram metrics for the PAM4 differential channel.

Metric	Simulated	Predicted
Bottom eye height	0.258	0.257
Bottom eye width (s)	5.38×10^{-11}	5.4×10^{-11}
Middle eye height	0.253	0.252
Middle eye width (s)	5.44×10^{-11}	5.43×10^{-11}
Top eye height	0.247	0.249
Top eye width (s)	5.39×10^{-11}	5.39×10^{-11}

substrate thickness = 10 mil, relative dielectric constant = 3.7, relative permeability = 1, conductor conductivity = 5.8e7 S/m, conductor thickness = 1.4 mil, and dielectric loss tangent = 0.002. V_{TX} is the output voltage of the transmitter (TX), V_{RX} is the input voltage to the receiver (RX), and V_{TX0} is the output voltage of TX when it is terminated with a 100 Ω resistor. The TX outputs a PRBS bit sequence with a rate of 14 GBd with a rise/fall time of 30 picoseconds. The sampling rate of the voltage waveforms is 10 samples per symbol. The modeling problem is defined in the same way as in the previous two examples. The CNN-LSTM model is generated from the ASH-HPO algorithm with the settings: $W_{LB} = 1600$, $W_{LA} = 5$, apply "both penalties", number of initial models = 81, $\lambda = 3$, initial $L_{max} = (3 \times L_{t0})$, acquisition function = EI, and a termination goal of validation loss $\leq$ 2e-5. A total of 72k training

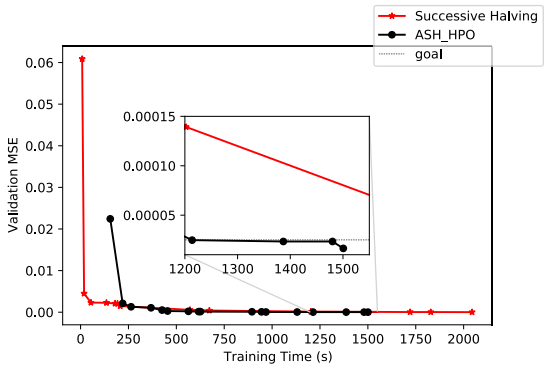

FIGURE 34. Validation losses of the ASH-HPO and successive halving algorithms versus time for the PAM4 differential channel. Note that the BO and hyperband algorithms both fail to reach the goal in this example.

data is used with a training time of 1500 seconds and a validation loss of 1.64e-5. Fig. 31 and Fig. 32 show the comparisons between the actual and predicted normalized V_{RX} of the PAM4 differential channel for time step, $t = 1$ to $t = 5\text{k}$ and $t = 400\text{k} + 1$ to $t = 405\text{k}$ of the testing dataset respectively. The prediction speed is about 310 microseconds per time step. No significant accuracy degradation is observed from the first to the last testing set. Fig. 33 shows the eye diagrams generated using the simulated and predicted voltage waveforms. Table 6 compares the eye diagram metrics for both eye diagrams. The ASH-HPO algorithm is also compared to other HPO algorithms in this example using the same 72k training data. Fig. 34 shows that the ASH-HPO algorithm converges faster than the successive halving algorithm, while the BO and hyperband algorithms both fail to reach the goal.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented the ASH-HPO algorithm for automated hyperparameter optimization of CNN-LSTM models for transient simulations and demonstrate it using three high-speed channels. It is shown that our proposed method converges faster than three state-of-the-art HPO methods: BO, successive halving, and hyperband algorithms. We investigated the different aspects of the algorithm and show that our algorithm can be further improved by applying two penalties, which punishes slow training models and bad models and their neighbors, and show that the method can converge with a similar speed even with different choices of initial lengths of training data. Although the ASH-HPO algorithm has its own parameters such as the W_{LA}, initial L_{max0}, choice of acquisition function, L_{v0}, and L_{t0}, these paramaters have a much less significant impact on the convergence rate of the algorithm, compared to the effect of the hyperparameters of the neural models.

For future research, the ASH-HPO algorithm can be used in combination with other advanced and recent models such as the LSTM with attention mechanism and the transformer model. Other than that, the high-speed channel modeling problem can be broken down into several smaller problems, where each problem focuses on the transient modeling of a certain part of the whole channel. Then, whenever that part of the channel is changed during the design process, another neural model can be swapped in or out. In this case, the ASH-HPO algorithm can be scaled up to select the combination of models that can successfully model the whole system. In this work, a rolling forecast method is employed for the training and prediction of the transient waveform. If the sequence to be predicted is indefinitely long, accumulation of errors can become an issue to the accuracy of the predictions. In such a case, the network can be retrained after a certain number of bits, and the presented ASH-HPO algorithm would be very well suited to simplify and automate the whole process. As the neural networks are deterministic in nature, only deterministic components are considered in this work. If necessary, a common practice is to add random components in a postprocessor as Gaussian distributions, and the modeling and subsequent hyperparameter optimization of these systems can be investigated as well. Finally, although the examples presented here focused on the transient simulations of high-speed channels, the developed algorithm is not limited to this field. Other applications in time series forecasting problems such as stock price prediction can be pursued as well.

REFERENCES

[1] T.-L. Wu, F. Buesink, and F. Canavero, "Overview of signal integrity and EMC design technologies on PCB: Fundamentals and latest progress," *IEEE Trans. Electromagn. Compat.*, vol. 55, no. 4, pp. 624–638, Aug. 2013.

[2] E. Sicard, "Recent advances in electromagnetic compatibility of 3D-ICs–Part I," *IEEE Electromagn. Compat. Mag.*, vol. 4, no. 4, pp. 79–89, 4th Quart., 2015.

[3] E. Sicard, W. Jianfei, R. Shen, E. P. Li, E.-X. Liu, J. Kim, and J. Cho, "Recent advances in electromagnetic compatibility of 3D-ICs—Part II," *IEEE Electromagn. Compat. Mag.*, vol. 5, no. 1, pp. 65–74, 1st Quart. 2016.

[4] S. H. Hall and H. L. Heck, *Advanced Signal Integrity for High-Speed Digital Designs*. Hoboken, NJ, USA: Wiley, 2011.

[5] B. K. Casper, M. Haycock, and R. Mooney, "An accurate and efficient analysis method for multi-Gb/s chip-to-chip signaling schemes," in *Proc. Symp. VLSI Circuits. Dig. Tech. Papers*, Jun. 2002, pp. 54–57.

[6] A. Sanders, M. Resso, and J. D. Ambrosia, *Channel Compliance Testing Using Novel Statistical Eye Methodology*. Santa Clara, CA, USA: DesignCon, 2004.

[7] M. Tsuk, D. Dvorscak, C. S. Ong, and J. White, "An electrical-level superposed-edge approach to statistical serial link simulation," in *IEEE/ACM Int. Conf. Comput.-Aided Design-Dig. Tech. Papers*, Nov. 2009, pp. 717–724.

[8] M. Ahadi Dolatsara, J. A. Hejase, W. D. Becker, and M. Swaminathan, "A hybrid methodology for jitter and eye estimation in high-speed serial channels using polynomial chaos surrogate models," *IEEE Access*, vol. 7, pp. 53629–53640, 2019.

[9] M. A. Dolatsara, J. A. Hejase, W. D. Becker, J. Kim, S. K. Lim, and M. Swaminathan, "Worst-case eye analysis of high-speed channels based on Bayesian optimization," *IEEE Trans. Electromagn. Compat.*, vol. 63, no. 1, pp. 246–258, Feb. 2021.

[10] T. Lu, K. Wu, Z. Yang, and J. Sun, "High-speed channel modeling with deep neural network for signal integrity analysis," in *Proc. IEEE 26th Conf. Elect. Perform. Electron. Packag. Syst. (EPEPS)*, San Jose, CA, USA, Oct. 2017, pp. 1–3.

[11] N. Ambasana, G. Anand, D. Gope, and B. Mutnury, "S-parameter and frequency identification method for ANN-based eye-height/width prediction," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 7, no. 5, pp. 698–709, May 2017.

[12] N. Ambasana, G. Anand, B. Mutnury, and D. Gope, "Eye height/width prediction from *S*-parameters using learning-based models," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 6, no. 6, pp. 873–885, Jun. 2016.

[13] 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.

[14] C. K. Ku, C. H. Goay, N. S. Ahmad, and P. Goh, "Jitter decomposition of high-speed data signals from jitter histograms with a pole–residue representation using multilayer perceptron neural networks," *IEEE Trans. Electromagn. Compat.*, vol. 62, no. 5, pp. 2227–2237, Oct. 2020.

[15] Y. Chu, F. Chen, J. Lang, and B. Lee, "Equalization with neural network circuitry for high-speed signal link," in *Proc. IEEE Int. Symp. Electromagn. Compat., Signal Power Integrity (EMC+SIPI)*, New Orleans, LA, USA, Jul. 2019, pp. 625–628.

[16] H. Kim, C. Sui, K. Cai, B. Sen, and J. Fan, "An efficient high-speed channel modeling method based on optimized design-of-experiment (DoE) for artificial neural network training," *IEEE Trans. Electromagn. Compat.*, vol. 60, no. 6, pp. 1648–1654, Dec. 2018.

[17] Z. Naghibi, S. A. Sadrossadat, and S. Safari, "Time-domain modeling of nonlinear circuits using deep recurrent neural network technique," *AEU-Int. J. Electron. Commun.*, vol. 100, pp. 66–74, Feb. 2019.

[18] B. O'Brien, J. Dooley, and T. Brazil, "RF power amplifier behavioral modeling using a globally recurrent neural network," in *IEEE MTT-S Int. Microw. Symp. Dig.*, San Francisco, CA, USA, Jun. 2006, pp. 1089–1092.

[19] Y. Cao and Q.-J. Zhang, "A new training approach for robust recurrent neural-network modeling of nonlinear circuits," *IEEE Trans. Microw. Theory Techn.*, vol. 57, no. 6, pp. 1539–1553, Jun. 2009.

[20] C. Zhang, S. Yan, Q.-J. Zhang, and J.-G. Ma, "Behavioral modeling of power amplifier with long term memory effects using recurrent neural networks," in *Proc. IEEE Int. Wireless Symp. (IWS)*, Beijing, China, Apr. 2013, pp. 1–4.

[21] Y. Fang, M. C. E. Yagoub, F. Wang, and Q.-J. Zhang, "A new macromodeling approach for nonlinear microwave circuits based on recurrent neural networks," *IEEE Trans. Microw. Theory Techn.*, vol. 48, no. 12, pp. 2335–2344, Dec. 2000.

[22] H. Sharma and Q. J. Zhang, "Transient electromagnetic modeling using recurrent neural networks," in *IEEE MTT-S Int. Microw. Symp. Dig.*, San Francisco, CA, USA, Jun. 2005, pp. 1597–1600.

[23] D. Luongvinh and Y. Kwon, "A fully recurrent neural network-based model for predicting spectral regrowth of 3G handset power amplifiers with memory effects," *IEEE Microw. Wireless Compon. Lett.*, vol. 16, no. 11, pp. 621–623, Nov. 2006.

[24] 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)*, San Jose, CA, USA, Oct. 2018, pp. 303–305.

[25] T. Nguyen, T. Lu, K. Wu, and J. Schutt-Aine, "Fast transient simulation of high-speed channels using recurrent neural network," 2019, *arXiv:1902.02627*. [Online]. Available: http://arxiv.org/abs/1902.02627

[26] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning* (The Springer Series on Challenges in Machine Learning), F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-05318-5_1.

[27] D. Lho, J. Park, H. Park, S. Park, S. Kim, H. Kang, S. Kim, G. Park, K. Son, and J. Kim, "Bayesian optimization of high-speed channel for signal integrity analysis," in *Proc. IEEE 28th Conf. Electr. Perform. Electron. Packag. Syst. (EPEPS)*, Montreal, QC, Canada, Oct. 2019, pp. 1–3.

[28] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," 2015, *arXiv:1502.07943*. [Online]. Available: http://arxiv.org/abs/1502.07943

[29] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," 2016, *arXiv:1603.06560*. [Online]. Available: http://arxiv.org/abs/1603.06560

[30] X. Zeng and G. Luo, "Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection," *Health Inf. Sci. Syst.*, vol. 5, no. 1, p. 2, Dec. 2017.

[31] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Brisbane, QLD, Australia, Apr. 2015, pp. 4580–4584.

[32] T.-Y. Kim and S.-B. Cho, "Predicting residential energy consumption using CNN-LSTM neural networks," *Energy*, vol. 182, pp. 72–81, Sep. 2019.

[33] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, and S. Guadarrama, "Long-term recurrent convolutional networks for visual recognition and description," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 677–691, Apr. 2017.

[34] X. Shi, Z. Chen, H. Wang, D. Y. Yeung, W. K. Wong, and W. C. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 802–810.

[35] T. Li, M. Hua, and X. Wu, "A hybrid CNN-LSTM model for forecasting particulate matter (PM2.5)," *IEEE Access*, vol. 8, pp. 26933–26940, 2020.

[36] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010, *arXiv:1012.2599*. [Online]. Available: http://arxiv.org/abs/1012.2599

[37] J. T. Wilson, F. Hutter, and M. Peter Deisenroth, "Maximizing acquisition functions for Bayesian optimization," 2018, *arXiv:1805.10196*. [Online]. Available: http://arxiv.org/abs/1805.10196

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[39] J. Jiménez and J. Ginebra, "PyGPGO: Bayesian optimization for Python," *J. Open Source Softw.*, vol. 2, no. 19, p. 431, Nov. 2017.

[40] Y. Yu, X. Si, C. Hu, and Z. Jianxun, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Comput.*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019.

[41] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Phys. D: Nonlinear Phenomena*, vol. 404, Mar. 2020, Art. no. 132306.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

CHAN HONG GOAY was born in Bukit Mertajam, Penang, Malaysia, in 1992. He received the B.Eng. and M.Eng. degrees from the School of Electrical and Electronic Engineering, Universiti Sains Malaysia, in 2016 and 2019, respectively, where he is currently pursuing the Ph.D. degree.

Since 2019, he has been a Research Assistant with the School of Electrical and Electronic Engineering, Universiti Sains Malaysia. His current research interests include artificial intelligence-based circuit modeling and signal integrity.

NUR SYAZREEN AHMAD (Member, IEEE) received the B.Eng. (Hons) degree in electrical and electronic engineering and the Ph.D. degree in control systems from The University of Manchester, U.K., in 2009 and 2012, respectively.

She is currently with the School of Electrical and Electronic Engineering, Universiti Sains Malaysia. Her current research interests include robust constrained control, intelligent control systems, computer-based control, and autonomous mobile systems in wireless sensor networks.

PATRICK GOH received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 2007, 2009, and 2012, respectively.

Since 2012, he has been with the School of Electrical and Electronic Engineering, Universiti Sains Malaysia, where he currently specializes in the study of signal integrity for high-speed digital designs. His research interest includes development of circuit simulation algorithms for computer-aided design tools. He was a recipient of the Raj Mittra Award, in 2012, and the Harold L. Olesen Award, in 2010. He has served on the technical program committee and international program committee for various IEEE and non-IEEE conferences around the world.