Received 19 August 2022; revised 15 October 2022; accepted 20 November 2022. Date of publication 24 November 2022;
date of current version 13 December 2022. The review of this article was arranged by Editor P. Pavan.

Digital Object Identifier 10.1109/JEDS.2022.3224433

Comprehensive Investigation and Comparative Analysis of Machine Learning-Based Small-Signal Modelling Techniques for GaN HEMTs

SADDAM HUSAIN [1], MOHAMMAD HASHMI [1] (Senior Member, IEEE), AND FADHEL M. GHANNOUCHI [2] (Fellow, IEEE)

1 Department of Electrical and Computer Engineering, School of Engineering and Digital Sciences, Nazarbayev University, Nur-Sultan 010000, Kazakhstan
2 Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada

CORRESPONDING AUTHOR: M. HASHMI (e-mail: mohammad.hashmi@nu.edu.kz)

This work was supported by the Collaborative Research Grant (CRP) at Nazarbayev University under Grant 021220CRP0222.

ABSTRACT A number of machine learning (ML) algorithm based small signal modeling of Gallium Nitride (GaN) High Electron Mobility Transistors (HEMTs) have been reported in literature. However, these techniques rarely provide any inkling about their suitability in modeling GaN HEMTs under varied operating conditions. In this context, this paper thoroughly investigates various ML based techniques and identifies their suitability for specific application scenarios. At first, an array of commonly employed modeling techniques based around Artificial Neural Network, RANdom SAmple Consensus, Support Vector Regression, Gaussian Process Regression, Decision Tree, and Genetic algorithm assisted Artificial Neural Network are used for development of modeling framework to exploit the bias, frequency and geometry dependence on S-parameter based outputs. Subsequently, the ensemble techniques namely Bootstrap aggregating, Random Forests, Extremely Randomized Trees, AdaBoost, Gradient Tree Boosting, Histogram-based Gradient Boosting, and Extreme Gradient Boosting are also explored to understand the capability of these algorithms in the development of GaN HEMT small signal models. Thereafter, an exhaustive analysis of bias and variance is carried out to figure out the most appropriate algorithms for specific applications. The discrepancies during model development are removed by tuning the hyperparameters of the respective models using Random search optimization with 5-fold cross validation technique. Post tuning, the models are evaluated in terms of generalization capability, Advanced Design System compatibility, computational efficiency, training and simulation time, models' capacity and parameters'tuning time.

INDEX TERMS ANN, DT, ensemble methods, GA-ANN, GaN HEMT, GPR, RSO, small-signal modelling and SVR.

I. INTRODUCTION

Last two decades have seen a rapid emergence of Gallium Nitride (GaN) HEMTs as the main device employed in the design of advanced radio frequency (RF) power amplifiers (PAs) for varied applications such as 5G, military, and space etc. [1], [2], [3], [4], [5], [6], [7], [8]. It is imperative to note that this wide acceptance of GaN HEMTs has happened due to the development of their innovative large-signal models (LSMs) which are compatible with computer aided design (CAD) tools. Interestingly, the small-signal models (SSMs) are precursor for the eventual development of effective of LSMs [9], [10], [11], [12], [13]. The Equivalent Circuit Models (ECMs) are the most sought-after methods for SSMs of GaN devices [14], [15], [16], [17], [18]. These are widely

adopted and used in both academic research and industry applications [8], [19]. ECMs often exploit direct extraction techniques, which famously make use of "cold" approach and hybrid approaches, which allude to amalgamation of direct techniques with optimization algorithms to extract the model parameters [17], [18], [19], [20], [21]. The effectiveness of these methods rely on the quality of the measurements, extraction procedures, optimization techniques and well-thought assumptions [21], [22]. ECMs, which are extracted from experimental measurements of the device, provide tangible understanding of the physics of the device consequently convey a better feedback to the circuit designers and simultaneously accurately simulate the S-parameters. However, because of the dynamic and strong non-linear behaviors possessed by GaN HEMTs, the parameter extraction in such methods turns out to be quite challenging, imprecise and computationally inefficient at higher frequencies. furthermore, selection of suitable topology of ECMs, physically inconsistent values for complex topologies are major concerns for the designers [9], [10], [23].

Therefore, the alternative machine learning (ML) based techniques to develop SSMs are getting traction [24], [25], [26], [27], [28], [30], [31], [32], [34], [35]. These techniques have shown promise as they can emulate complex behaviors, manifest better prediction capability and generally are computationally efficient, nevertheless, requisite large sample size of the measurements [23], [24]. But there still exist many unresolved mysteries about these approaches because a number of aspects have not been explored as yet. In general, there is no generalized ML based modeling technique which can be called optimal for wide variety of applications. In fact, the efficacy of ML algorithms depends on the specific problem, quality of the data, feature extraction, nature of the data, distributions of the outputs and inputs, number of samples, division of the data, preprocessing techniques, type of the ML algorithm, parameter tuning, algorithms to tune the parameters, error functions and selected performance metrics. The selection of ML algorithms for model development is also contingent on the final design application. For instance, some application may require trade-off between computational efficiency and accuracy and hence appropriate ML algorithms need to be chosen accordingly. Furthermore, if the intended application requires unique model then choice of ML algorithms which produce unique solutions take precedence. A quick literature survey reveals that, within the broad ML based techniques, the Artificial Neural Network (ANN) [24], [25], [26], Support Vector Regression (SVR) [27], [28], Gaussian Process Regression (GPR) [30], [31], Decision Tree (DT) [32], and global optimization aproach [34], [35], [36] are frequently used for the small signal modelling of GaN HEMTs. However, each of these modelling techniques are often apt for some specific set of device operating conditions and behave poorly for the other operating conditions. So far there is almost no information about the generalization capability of ML based modeling of GaN HEMTs and this often leads to a scenario where designers and researchers are often unaware about the capabilities of ML based modeling techniques.

Therefore, it is envisaged that a thorough comparative analysis of the commonly reported ML based GaN HEMT small signal modeling techniques will bring a dual pronged benefits. It will become a one-stop information resource in the domain of ML based small signal modeling and at the same time it will also pave the way for further advancement in the modeling state-of-the-art. To facilitate these assertions, this paper thoroughly investigates and revisits ANN, Genetic algorithm (GA) initialized ANN, Random Search Optimization (RSO)-tuned RANdom SAmple Consensus (RANSAC), RSO-tuned SVR, RSO-tuned GPR and RSO-tuned DT for the behavioral modelling of GaN HEMTs devices. Furthermore, this paper also explores an array of ensemble techniques namely the Bootstrap Aggregating model (BAM), Random Forest (RFs), Extremely Randomized Trees (ERTs), Boosting methods such as AdaBoost with DT, AdaBoost with RFs, Gradient Tree Boosting (GB), Histogram based Gradient Boosting (HGB) and Extreme Gradient Boosting models (XGBoost) to identify the most effective and appropriate ML algorithm to model specified GaN HEMTs devices. The models obtained from all these ML techniques are then examined for various metrics such as computational efficiency, training and simulation time, models' capacity and parameters' tuning time, and their generalization capability in terms of Mean Squared Error (MSE), Mean Absolute Error (MAE) and coefficient of determination (CoD) also known as R^2 on the unseen testing set.

Finally, this paper utilizes measurement data of two distinct GaN HEMTs devices to train and test the models. That enables us to understand the scalability of the respective ML based models. In summary, the main contributions of this paper are: (i) a thorough and systematic approach for the development of ANN, GA-ANN, SVR, GPR, RANSAC, DT and an array of ensemble techniques based small-signal modelling of GaN HEMTs, (ii) demonstration of hyperparameters tuning using RSO with 5-fold CV technique, and (iii) a detailed inspection of the models in terms of computational efficiency, generalization capability, scalability, training and simulation time, model's capacity, tuning time and dataset compatibility. Sections I and II include the description of the data and methodology used in this paper. Section III presents the theoretical and practical paradigm for the all the explored modeling techniques. Moving forward, model validation, and a discussion on the experimental results are given in Sections IV and V. Section VI concludes the paper.

II. DESCRIPTION OF THE DATASET AND METHODOLOGY

A. THE DATASET

The measurement data used in this paper has been provided by the defense research development organization (DRDO). Not much information about the devices, grown on silicon

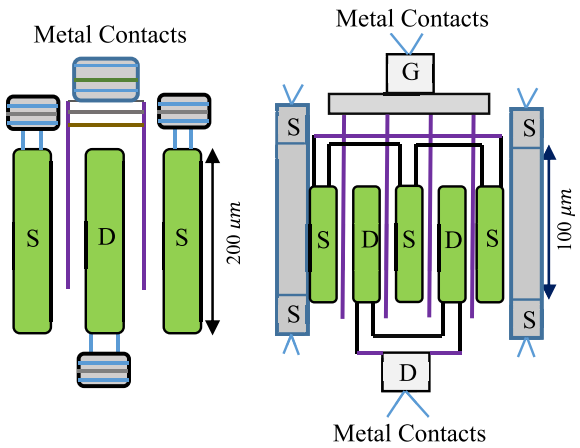

FIGURE 1. Layout of (left) 2x200 μm (right) 4x100 μm GaN-on-SiC HEMT devices.

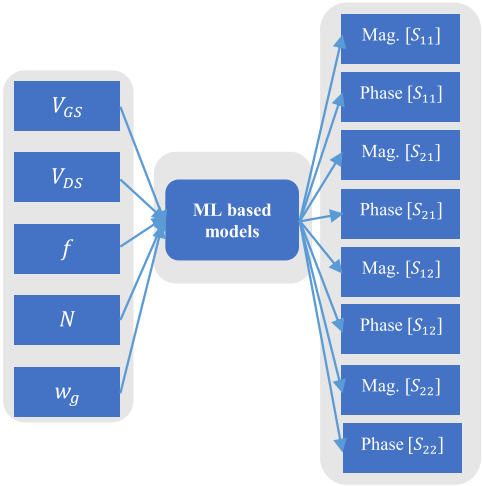

FIGURE 2. A generic diagram of proposed ML-based models.

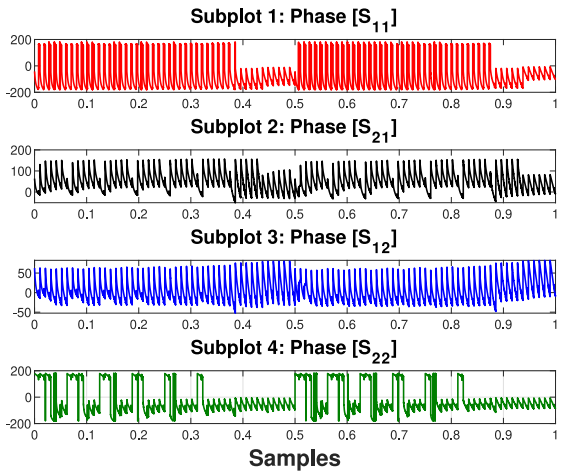

FIGURE 3. Distribution of the phase of the S-parameters.

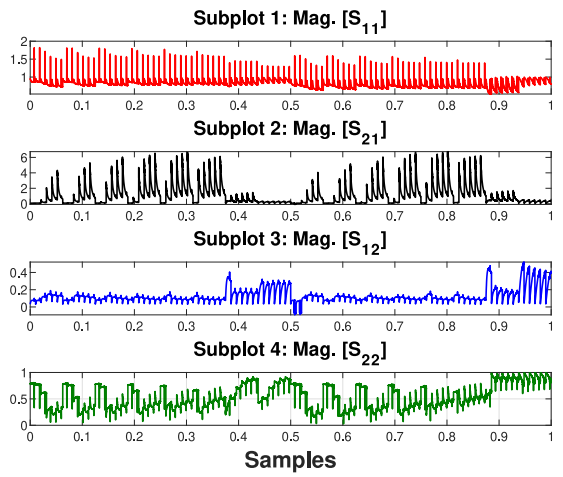

FIGURE 4. Distribution of the magnitude of the S-parameters.

carbide (SiC), can be shared due to sensitivities involved. However, it is imperative to mention that the measurement data are from GaN HEMT devices of geometries 2x200 μm and 4x100 μm as depicted in Fig. 1. Each device is biased following the standard biasing technique [1], and then their respective responses for varied bias conditions are recorded as S-parameters, using VNA, in terms of magnitude and phase. For both the devices, the gate to source voltage (V_{GS}) is varied from -7 V to 0 V with a step size of 1 V. Furthermore, the drain to source voltage (V_{DS}) is swept from 0 V to 10 V with a step size of 2 V. The device is characterized for the frequency (f) range of 1 to 18 GHz with a step size of 0.09 GHz. The complete dataset utilized in this paper is constructed by combining the samples for each device. The combined dataset consists of 17280 samples and embodies five predictor variables namely V_{GS}, V_{DS}, f, number of fingers ($N = 2, 4$) and unity gain width ($w_g = 200$ μm,100 μm). In addition, there are 8 predicted variables namely magnitude (mag.) S_{11}, phase S_{11}, mag. S_{21}, phase S_{21}, mag. S_{12}, phase S_{12}, mag. S_{22}, and phase S_{22} as shown in Fig. 2.

B. DATA PROCESSING METHODOLOGY

It is pertinent to understand the distribution of the dataset. Figs. 3 and 4 present the distribution of the phase and magnitude of each S-parameters for the full dataset. It can be observed that the ranges of most parameters are different. Furthermore, the output values at some samples are discontinuous and doesn't follow the overall distribution of the respective S-parameters. In addition, the effects of the outliers and uneven peak values are also evident and all these can be attributed to some anomaly in measurements. The obvious problem of having different ranges for the predictors and predicted variables is the orientation of the contour of the error function that may become skewed or distorted elliptical shape. This leads to erroneous results as the algorithm finds it hard to determine the global minima and instead converge at local minima in worst cases. Moreover, the outliers directly affect the placement of the decision boundary and this in turn give rise to a decision boundary that results in higher error on the testing set. Different algorithms behave

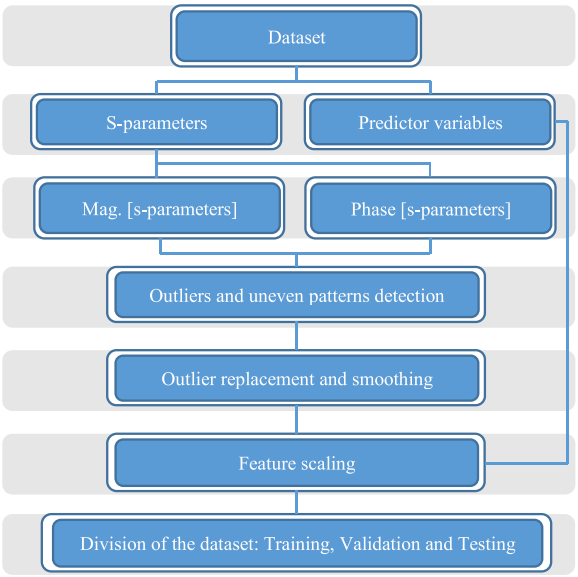

FIGURE 5. Preprocessing steps taken before training the models.

differently on dataset containing outliers. For instance, the mean absolute error function is less affected due to outliers than the mean squared error function. That is why it is essential to remove the outliers and smoothen out the peak values as much as possible. However, some outliers can not be removed because that may result in the loss of information. Another aspect is the division of the dataset. It is a standard practice to divide the data into training, validation, and testing sets. The training set is used to train the model, whereas the validation set helps in defining the hyperparameters. Finally, the independent testing set is used to compute the accuracy of the model.

The flow chart in Fig. 5 outlines the basic steps employed in the preprocessing stage. The magnitude and phase of the S-parameters at the preprocessing stage is cleaned through outlier detection and smoothing techniques. MATLAB programming language is used to identify the outliers and uneven peaks. We followed two-step process to identify outliers: firstly, we exploited *isoutlier* command in MATLAB with mean as a criteria; secondly, we plotted the measurements at all biasing conditions. Thereafter, we pinpointed the locations of the most outliers and replaced them with the mean of nearest values to avoid any loss of information. Similarly, the peaks are identified and smoothened. However, some of the outliers could not be removed as it was closely related with the overall behavior of the devices. Then all the predictor and predicted variables are arranged into numerical matrix form and scaled to almost similar range by dividing each variable with its respective absolute maximum value. Subsequently, the dataset is divided into training, validation and testing sets. For each V_{GS} value, 5 and 1 bias conditions (V_{DS}) are included in the training and the testing sets respectively. Furthermore, the bias conditions for training and testing datasets are chosen randomly to avoid any specific learning of the algorithm.

$$\text{MSE} = \frac{1}{n_{samples}} \sum_{j=1}^{n_{samples}} \left(y_j^{meas} - y_j^{pred}\right)^2 \tag{1}$$

$$\text{MAE} = \frac{1}{n_{samples}} \sum_{j=1}^{n_{samples}} |y_j^{meas} - y_j^{pred}| \tag{2}$$

$$R^2 = 1 - \frac{\sum_{j=1}^{n_{samples}} \left(y_j^{meas} - y_j^{pred}\right)^2}{\sum_{j=1}^{n_{samples}} \left(y_j^{meas} - \bar{y}\right)^2} \tag{3}$$

where $\bar{y} = \frac{1}{n_{samples}} \sum_{j=1}^{n_{samples}} y_j^{meas}$

Finally, the validation set is prepared using 5-fold cross validation (CV) scheme from the training set. For the sake of clarity, it is to be noted that only one fold eventually works as a validation fold out of the 5-folds of the training set. To tune the hyperparameters, the 5-fold CV with RSO [37] is utilized. The model development and investigations make use of these optimal hyperparameters. Thereafter, these optimal models are trained with complete training set and the predictions are recorded on both the training and unseen testing sets. The computation metrics MSE, MAE, and R^2 given by 1, 2 and 3 respectively are used to assess the generalization capability of the model, i.e., the accuracy of the trained ML models on independent unseen testing set.

III. MODEL DEVELOPMENT FRAMEWORK

A. ANN AND GA INITIALIZED MODELS

The detailed theoretical framework of ANN algorithm is well-known and extensively available in [11], [34], [35]. An essential aspect in the use of ANN for the model development is the selection of topology (i.e., the number of predictor variables, the number of unit neurons, the number of hidden layers of the model, activation functions, number of output layers, etc.) for a given dataset. The solution adopted in this paper makes use of trial-and-error approach. Initially a simple topology (a topology of one hidden layer and ten unit neurons in it) is selected. Then, 5-fold CV approach is utilized to compute the MSE, MAE and R^2 for the training, validation and testing sets. Thereafter, the model's capacity is gradually enlarged until the model is overfitted as at this point it renders the extreme topology. After getting to the overfitted topology, the model's capacity is slowly reduced until it provides a better fitting. Once the optimal topology is identified, this becomes the choice for getting the predictions on the training and testing data sets. This process is repeated for each S-parameter.

One hidden layer with sufficient neurons can approximate any mathematical relationship [39]. However, the increase in the number of hidden layers with less neurons in each layer can yield better performance [39]. This is due to the characteristic that the deeper the layer the more information can be fetched by unit neurons in each layer. Interestingly, going more deeper reduces the dependency of the model's performance on feature extraction. With this perspective,

deeper layers are adopted in this paper. Here, the model utilizes five inputs V_{GS}, V_{DS}, f, N, and w_g while the outputs are S-parameters (see Figure 2). Eight distinct models are developed in total as follows:

- Model's topology for mag. [S_{11}]: 5-10-10-10-1 (Number of input nodes- neurons in the first hidden layer-neurons in the second hidden layer- neurons in the third hidden layer- number of output nodes)
- Model's topology for phase [S_{11}]: 5-10-10-10-1
- Model's topology for mag. [S_{21}]: 5-12-12-12-1
- Model's topology for phase [S_{21}]: 5-12-12-12-1
- Model's topology for mag. [S_{12}]: 5-12-12-12-1
- Model's topology for phase [S_{12}]: 5-12-12-12-1
- Model's topology for mag. [S_{22}]: 5-10-10-10-1
- Model's topology for phase [S_{22}]: 5-10-10-10-1

The analytical learning equations for ANN models are given in (4)-(7). The first layer serves as input layer, the second, third, and fourth layers are hidden layers, and the fifth layer acts as the output. The topology consists k, m and n neurons in first, second and third hidden layers respectively. Here, $h_i^{(2)}$, $h_j^{(3)}$, $h_l^{(4)}$ and S-parameters [mag./phase] are the outputs at the second, third, fourth and fifth layers respectively. The tanh is the tan-sigmoid activation functions. $w_{i1}^{(1)}$, $w_{i2}^{(1)}$, $w_{i3}^{(1)}$, $w_{i4}^{(1)}$ and $w_{i5}^{(1)}$ are the weights for connections joining the input nodes to second hidden layer. Similiarly, $w_{ji}^{(2)}$, $w_{lj}^{(3)}$ and $w_l^{(4)}$ are the weight connections from the second to third, third to fourth and fourth to fifth layers respectively. $b_i^{(2)}$, $b_j^{(3)}$, $b_l^{(4)}$ and $b_y^{(5)}$ are the biases given to the respective neurons.

$$h_i^{(2)} = \sum_{i=1}^{k} \tanh\Big(b_i^{(2)} + w_{i1}^{(1)} \times V_{GS} + w_{i2}^{(1)} \times V_{DS} + w_{i3}^{(1)} \times f + w_{i4}^{(1)} \times N + w_{i5}^{(1)} \times w_g\Big) \tag{4}$$

$$h_j^{(3)} = \sum_{j=1}^{m} \tanh\left(b_j^{(3)} + \sum_{i=1}^{k} w_{ji}^{(2)} \times h_i^{(2)}\right) \tag{5}$$

$$h_l^{(4)} = \sum_{l=1}^{n} \tanh\left(b_l^{(4)} + \sum_{j=1}^{m} w_{lj}^{(3)} \times h_j^{(3)}\right) \tag{6}$$

$$\text{S-parameters [mag./phase]} = b_y^{(5)} + \sum_{l=1}^{n} w_l^{(4)} \times h_l^{(4)} \tag{7}$$

Use of deeper neural layers may give rise to vanishing gradient problem which is detrimental for the algorithm. Moreover, the uniqueness of the solutions is one of the requirements for categorizing the efficiency of the model during the design of circuits such as power amplifiers [26]. It is observed that the backpropagation (BP) algorithms, initially, generate their weights and biases randomly. It is quite possible that due to inherent ANN feature the generated solutions could be driven to the local minimums as the updated weights depend on the initial weights. Also, depending on the locations of the initial weights, the algorithm will follow the nearest steepest descent path to find the convergence. The reproducibility of the solutions is also a major concern with the BP-based algorithms. Therefore, we incorporate GA in this paper to overcome this problem associated with local minimums. The GA is chosen as it possesses strong exploration capability. Depending on the number of populations generated initially it can cover a large search area and the solutions can guarantee the convergence of the model at global minimums [11]. GA being the evolutionary algorithm, influenced by the natural selection, commence by generating an initial population that goes through fitting, selection, recombination, mutations and reinsertion steps in order to produce the offspring for the next iteration. More details of GA algorithms are given in [11], [21], [34], [35], [40]. The flow chart in Fig. 6 outlines the steps taken to build the GA initialized ANN model, while the steps to develop the model are described below.

- The number of individuals directly influence the simulation time and computational efficiency. Therefore, GA commences by generating the targeted number of individuals which act like a hyperparameter. In the present study, initial populations of 100, 500, 1000, 1500 and 3500 individuals are generated and a trial and error analysis is conducted. The population of 100 individuals could not find the good initial weights, a packet of 500 individuals also gave inferior weights than the 1000 individuals based generation. The performance for 1000 and 1500 individuals are comparable whereas 3500 individuals produced slightly better performance but at the cost of significantly increased simulation time. As a trade-off between the performance and simulation time, the population containing 1000 individuals are selected for the mag/phase of S_{11}, S_{22}, and population of 1500 individuals for mag/phase of S_{21}, S_{12}. Each individual contains the total number of weights and the biases are initiated randomly between -1 and 1. The termination condition is set by defining the number of iterations, also a tunable parameter, the GA algorithm will run. The results are tested with 250, 500, 750, 1000 and 1500 iterations. 1000 iteration is found to be sufficient and provide a good trade-off between performance and simulation time.
- The next step involves the computation of the fitness score. The MSE loss function is exploited for this task.
- The best fitted individuals go through the process of selection, recombination and mutation stages after the calculation of the fitness score. In order for them to produce the offspring using the highly efficient double-edge crossover technique, a certain percentage of less fitted variables are excluded. The percentage of individuals to be excluded is set by setting the generation gap, a parameter, that requires tuning. The generation gap are set at 0.95, 0.9, 0.8, 0.7 and 0.5. Here, the best performance is observed when setting the generation gap to 0.9, i.e., 10% of the populations are excluded at each iteration.

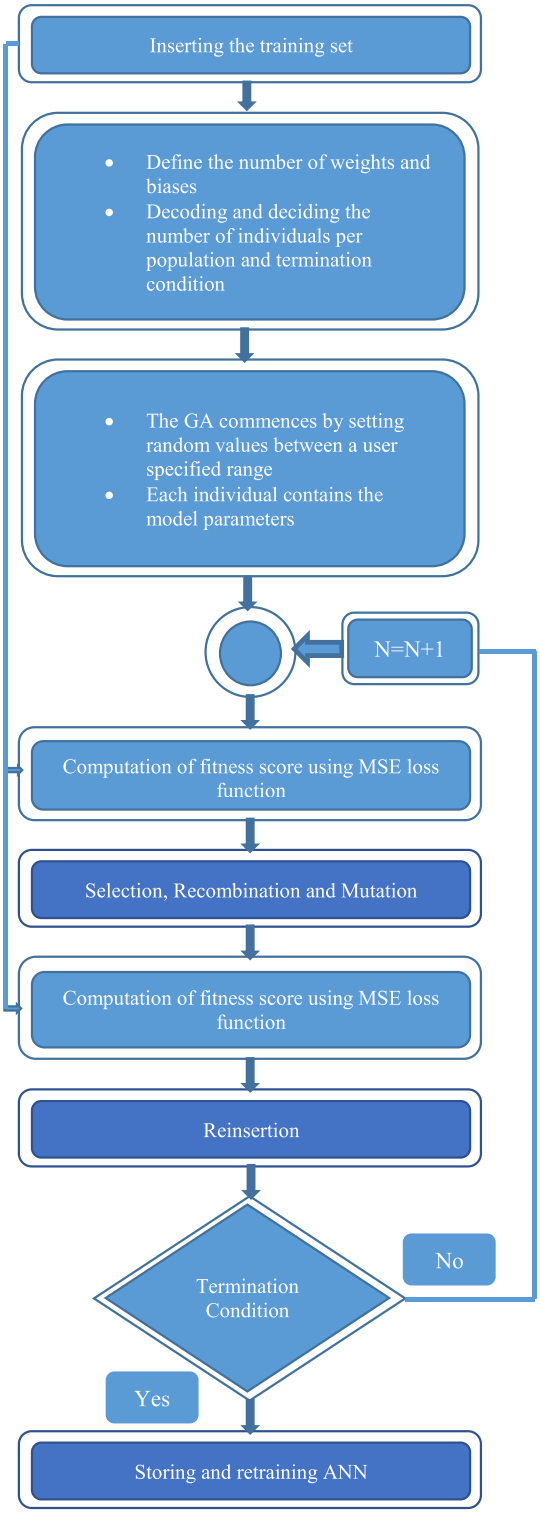

FIGURE 6. The flow chart explaining the GA-ANN model's training procedure.

- Finally the termination condition is checked. One of the conditions of the termination is iteration number, while the other condition is the relative change in the error. Once the termination condition is met, the algorithm generates a set of weights which are used as initial weights for the ANN.

B. RANDOM SEARCH OPTIMIZATION-TUNED RANSAC MODEL

This model is selected due to its well-known robust performance on corrupt data or data with outliers and irregular distribution [41]. The idea is to separate the outliers from the complete dataset and then train the model on the remaining data. The RANSAC makes use of only the subset of inliers, i.e., the cluster of samples distinct from the outliers to fit the model. The fitted regressor is then evaluated on the training and testing sets. Due to the non-deterministic nature, its performance is dependent on the number of iterations. As mentioned earlier, the hyperparameters are tuned with RSO using 5-fold CV technique. Here, the hyperparameters are base estimator, maximum number of iterations and loss functions. The base estimator employs linear regression and polynomial regression as its two variants. The maximum iteration is set to take values from the range of [10, 2000]. Lastly, the tuning is obtained for MSE and MAE loss functions. Once the tuned parameters are acquired, the model is again trained and tested.

C. RANDOM SEARCH OPTIMIZATION-TUNED SVR MODEL

SVR is a non-parametric kernel-based approach to approximate the regression-based problems. It embodies absolute error loss function that makes it a right choice for outliers ingrained dataset. The ultimate objective of SVR algorithm is to try to find a decision boundary or the regression line that deviates from the true output for at most ϵ, for each training inputs, and maintain the generalization capability as far as possible. In pursuit of these objectives, it enacts linear epsilon-insensitive SVM (ϵ-SVM), otherwise called $L1$ loss. The mathematical derivation of SVR algorithm has been comprehensively discussed in the literature [11], [27], [29], [35]. If $f(x)$ constitute the decision boundary or function for a training set of multivariate set of inputs (x_n) of N observations and its corresponding outputs (y_n), the function $f(x)$ can be represented in terms of the support vectors [42]. Here, α_n and $\alpha_n^\star$ denote the weight coefficients of the support vectors, x is the multivariate set of inputs, $G(.)$ is the gram matrix, and b is the bias term.

$$f(x) = \sum_{i=1}^{N} (\alpha_n - \alpha_n^\star) G(x_n, x) + b \tag{8}$$

In order to derive the prediction function, suppose x is the multivariate set of testing inputs given as $x = [V_{GS}, V_{DS}, f, N, w_g]$, and y_s is the corresponding S-parameter. Similarly, x_n and y_n can be defined from the training set. Using (8), the predictive function $f(x)$ is given by (9). Here, t_n denotes the support vectors, N the number for support vectors and σ the Gaussian

parameter.

$$\text{Mag./Phase}[S_{ij}] = \sum_{i=1}^{N}(\alpha_n - \alpha_n^{\star}) \exp\left(\frac{(t_n - \{[V_{GS}, V_{DS}, f, N, w_g]\})^T (t_n - \{[V_{GS}, V_{DS}, f, N, w_g]\})}{2\sigma^2}\right) + b \tag{9}$$

To acquire the best out of the SVR algorithm, the parameters are tuned using RSO with 5-fold CV for 250 iterations. Before the tuning, first the same preprocessing steps are given to the dataset, followed by the tuning of Kernel function, Box constraint (C), ϵ, and kernel scale. The kernel functions can be set to Linear, Gaussian, polynomial kernels or a custom kernel. The ranges of C and ϵ are set to [1e-3, 1e3] and [1e-5, 1e5] respectively.

D. RANDOM SEARCH OPTIMIZATION-TUNED GPR MODEL

It is a non-parametric probabilistic approach for regression-based problems and can be used for device modeling. Unlike other models considered in this paper, it can directly predict on the testing set given the training set. The mathematical and theoretical constructs of GPR algorithm have been well-established and can be accessed through [11], [30], [43]. However, to explain briefly, how GPR predicts, suppose that the training set is given as $T_r = \{x_i, y_i\}$, where x_i is the multivariate set of inputs $x_i = \{[V_{DS}^i, V_{GS}^i, f^i, N^i, w_g^i]\}$; $i = 1, 2, 3, \ldots, N$, and y_i is the corresponding magnitude or phase S-parameters, i.e., $y_i = \text{mag./phase}[S_{ij}]$. Similarly, assuming that the testing set is given as $T_{test} = \{x_j^{test}, y_j^{test}\}$, where $x_j^{test} = \{[V_{DS}^j, V_{GS}^j, f^j, N^j, w_g^j]\}$; $j = 1, 2, 3, \ldots, n$ samples drawn from the testing set. The GPR commences by assuming a Gaussian process apriori by exploiting the mean $m(x)$ and covariance $k(x, x')$ as given in 10. In order to define the function, it computes the covariance matrix values between each input elements and then samples the function f_{sample}, given by (11), in terms of normal distribution. Generally it is a common practice to assume the mean to be zero. From the standard results, the instances of the conditional distribution $p(f_{sample}|T_r, x_j^{test})$ can be modeled using (12) and (13). Finally, the predicted S-parameters can be modeled using (14).

$$f(x) \sim GP\big(m(x), k\big(x, x'\big)\big) \tag{10}$$

$$f_{sample} \sim \mathcal{N}\left(0, k\left(x_j^{test}, x_j^{test}\right)\right) \tag{11}$$

$$m\left(x_j^{test}\right) = k\left(x_j^{test}, x_i\right)\left[k(x_i, x_i) + \sigma_m^2 I\right]^{-1} y_i \tag{12}$$

$$k_t\left(x_j^{test}, \left(x_j^{test}\right)'\right) = k\left(x_j^{test}, \left(x_j^{test}\right)'\right) - k\left(x_j^{test}, x_i\right) \left[k(x_i, x_i) + \sigma_m^2 I\right]^{-1} k\left(x_i, \left(x_j^{test}\right)'\right) \tag{13}$$

$$\text{mag./phase}\left[S_{ij}^{test}\right] \sim \mathcal{N}\left(m\left(x_j^{test}\right), k_t\left(x_j^{test}, \left(x_j^{test}\right)'\right)\right) \tag{14}$$

The hyperparameters in GPR are the basis function, kernel functions, kernel scale, sigma and standardization of the data and here these are optimized using RSO with 5-fold CV optimization technique. The termination condition is the iteration number which is 30 for each S-parameter. The basis function is determined by the number of samples. The kernel or covariance functions can be selected from exponential, squared exponential, matern kernel with parameter 3/2, matern kernel with parameter 5/2, rational quadratic kernels and their variants with separate length scale per predictor. It is pertinent to tune the parameter of the kernel as well. The range of the tunable parameters has to be chosen wisely. We started with a larger range—computed the values of the hyperparameters after running the optimization—examined the generalization of the tuned model—narrowed the range and repeated the process until the optimal values for the parameters is achieved.

E. RANDOM SEARCH OPTIMIZATION-TUNED DECISION-TREE MODEL

DT is a non-parametric supervised learning-based algorithms, designed to be operated for both regression and classification. It makes use of simple if-then-else decision rules to derive the decision boundary. Similar to other ML algorithms discussed in this paper, its goal is to obtain an optimal decision boundary for the presented training set. In its basic configuration, it is analogous to a tree like structure, incorporating root, branches, internal and leaf nodes [44]. The algorithm embarks from the root node, and goes through some intermediate nodes. The number and layers of intermediate nodes depend on the complexity of the problem. Finally, it terminates at the leaf nodes that contain the predicted values [32], [33]. They are easy to be use due to the simplistic structure. However, development of overly-complex DT models are difficult as it requires tuning of the hyperparameters. Here, the search for optimal parameters is carried out for 500 iterations using RSO with 5-fold CV. The tuned parameters are the maximum depth of the tree (max depth) [1 to 1000 with a step size of 1], minimum samples per leaf (min samples leaf) [1 to 200 with a step size of 1], minimum number of samples vital to split the internal node (min samples split) [2 to 200 with a step size of 1], maximum number of features (max features) [2 to 10 with a step size of 1] and criterion (the function to measure the quality of the split): [squared error, absolute error].

F. RANDOM SEARCH OPTIMIZATION-TUNED ENSEMBLE MODELS

It has been identified that the combinations of different ML approximators may result in the determination of better generalization over the given problem. Ensemble is an approach in this direction. The objective is to amalgamate prediction abilities of some base estimators in pursuit of better generalization. The ensemble includes two families of methods namely the averaging methods, where the idea is to build independent estimators and then averaging their predictions,

and the boosting methods, where the base algorithms work sequentially. All the ensemble models developed in this paper are tuned using RSO with 5-fold CV technique.

F.1. BOOTSTRAP AGGREGATING MODEL (BAM)

It is a meta-estimator that fits the base estimator to randomly selected (sampling with replacement) subsets from the original dataset and computes the performance on each subset before clustering them together in order to provide the final prediction [45]. It is used to remove the variance of the base estimator and in general this model has proven to produce better results in overfitting cases. Here the DT is chosen as the base estimator. The tuning of the numbers of the base estimators (between 1 to 1000 with a step size of 10) produced satisfactory results.

F.2. RANDOM FORESTS (RFS) AND EXTREMELY RANDOMIZED TREES (ERTS) MODELS

These are special cases designed to ameliorate the performance of DTs. These are averaging algorithms based on randomized DT. For RFs, two layers of randomness in inserted to improve the generalization. Initially, each tree in RFs model is created from the sample drawn with replacement from the training set. Finally, while splitting each node for a tree, the algorithm tries to find the best split either from all input vectors or using the parameter maximum number of features [46]. However, ERTs use an extra layer of randomization. The extra layers enable the algorithms to reduce the variance at the cost of minute rise in bias [47]. The tunable-parameters for RFs and ERTs are the number of trees (between 1 to 1000 with a step size of 1), the maximum depth of the trees (between 1 to 200 with a step size of 1), the samples needed to split the internal nodes (between 1 to 200 with a step size of 1), samples needed to be at a leaf node and maximum number of features for the best split (between 1 to 10 with a step of 1).

F.3. ADABOOST, GRADIENT TREE BOOSTING AND HISTOGRAM BASED GRADIENT BOOSTING (HGB) MODELS

Boosting models are a possible substitute for the bagging models. They primarily focus on altering the weak learners to strong learners by working with each model where their performance is not good. An AdaBoost regressor works on the principle of fitting the original regressor and extra copies of regressor on the data. It amends the data and sample weights at each iteration according to the error values [48]. The performance of AdaBoost is greatly influenced by the base estimator, the number of estimators to embark the algorithm and the learning rate. A trade-off for these parameters are necessary to get the optimal model for any given problem. Two models are developed to demonstrate these aspects. The first using DT as the base estimator and the second using the RFs as the base estimator. The number of estimators are tuned between 1 to 1000 with a step size of 1 with the learning rate that is transverse in the range of 0.1 to 5 with a step size of 0.1. Gradient boosting (GB) constructs an added substance model in a forward stage-wise style; it considers the improvement of arbitrary differentiable loss functions. During each iteration, it fits an RT on the negative gradient of the loss functions. Furthermore, the extension of GB is HGB as they are faster and produce better results for a dataset containing many number of samples [49]. The parameters that are tuned for HGB models are the loss functions (least square, least absolute, and Huber loss functions), the maximum number of iterations (between 1 to 1000 with a step size of 1), the maximum number leaves for each tree (between 1 to 200 with a step size of 1), and the minimum number of leafs (between 1 to 200 with a step size of 1). The same parameters with the same range are also tuned for GB model except that the number of iterations are replaced with number of estimators. In this paper, the AdaBoost, GB, and HGB are tuned for 250 iterations whereas AdaBoost with RFs are tuned for 50 iterations.

F.4. EXTREME GRADIENT BOOSTING MODEL (XGBOOST)

It is also a gradient boosting method with some strong upgrades. It is fast due to the incorporation of methods like parallel computing, and produce an accurate and less overfitting results. In its basic configuration it includes a unique split algorithm alongside the regularization term to control the overfitting [50]. The model has a unique set of hyperparameters which must be fine-tuned. The main parameters tuned here are learning rate (between 0.1 to 10 with step size of 1), maximum depth (between 1 to 1000 with step size of 1) and the number of estimators (between 1 to 10000 with step size of 10).

IV. MODEL VALIDATION AND EVALUATION

This section presents a detailed validation and evaluation results of the models considered in this paper. It is important to mention that MATLAB and Python are used for the development, validation, and evaluation of the models.

A. EVALUATION OF ANN AND GA INITIALIZED MODELS

The ANN models are developed in MATLAB and are then trained using Levenberg-Marquardt backpropagation [51]. Other backpropagation-based training algorithms such as gradient descent, gradient descent with momentum, variable learning rate gradient descent, BFGS Quasi-Newton do not produce satisfactory results. The scaled Conjugate Gradient and Resilient Backpropagation algorithms did produce comparable results but on every metric the Levenberg-Marquardt algorithm produces slightly superior result. Furthermore, the Bayesian regularization backpropagation produces the most accurate results, but at the cost of higher simulation time, as it embodies automatic regularization that enables the algorithm to work in overfitting cases. Therefore the Levenberg-Marquardt backpropagation is chosen in this work considering the trade-off between simulation time and accuracy. The parameters are tuned with 5-fold CV technique, and

TABLE 1. Outcomes of the ANN models on the independent testing set (Generalization).

Metrics	Bias	Mag $[S_{11}]$	Phase $[S_{11}]$	Mag $[S_{21}]$	Phase $[S_{21}]$	Mag $[S_{12}]$	Phase $[S_{12}]$	Mag $[S_{22}]$	Phase $[S_{22}]$
MSE	All bias	1.78e-5	1.32e-3	2.54e-4	2.25e-4	1.14e-4	3.12e-3	1.92e-4	2.94e-3
MAE	All bias	3.74e-3	1.71e-2	1.91e-3	2.78e-3	1.18e-3	2.66e-2	3.58e-3	3.19e-2
% R-Squared	All bias	99.31	99.12	99.38	99.17	99.26	99.32	99.18	98.85

TABLE 2. Outcomes of the GA initialized ANN models on the independent testing set (Generalization).

Metrics	Bias	Mag $[S_{11}]$	Phase $[S_{11}]$	Mag $[S_{21}]$	Phase $[S_{21}]$	Mag $[S_{12}]$	Phase $[S_{12}]$	Mag $[S_{22}]$	Phase $[S_{22}]$
MSE	All bias	6.22e-6	1.06e-3	1.12e-4	1.29e-4	1.09e-4	8.48e-4	1.57e-4	1.55e-3
MAE	All bias	9.84e-4	9.91e-3	6.97e-3	4.78e-3	7.18e-3	8.91e-3	5.58e-3	7.19e-3
% R-Squared	All bias	99.54	99.32	99.41	99.44	99.67	99.63	99.25	99.03

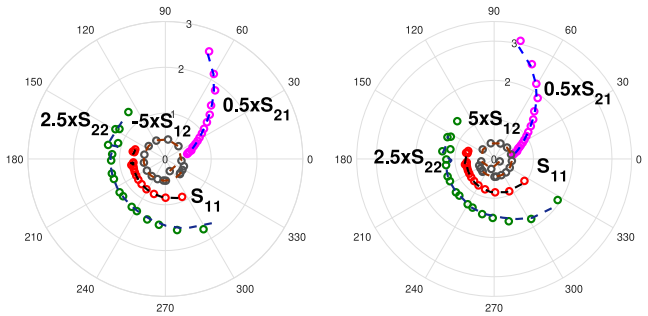

FIGURE 7. Measured (symbols) and predicted (dashed-lines) S-parameters at (a) $V_{GS} = -5$ V and $V_{DS} = 6$ V (b) $V_{GS} = -5$ V and $V_{DS} = 8$ V for two fingers GaN-on-SiC of 200 μm gate length each (for ANN models).

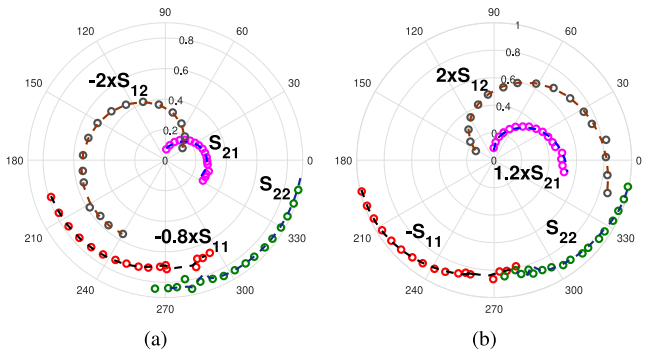

FIGURE 8. Measured (symbols) and predicted (dashed-lines) S-parameters at (a) $V_{GS} = -7$ V and $V_{DS} = 8$ V for four fingers GaN-on-SiC of 100 μm gate length each (b) $V_{GS} = -7$ V and $V_{DS} = 6$ V for two fingers GaN-on-SiC of 200 μm gate length each (for ANN models).

the MSE, MAE, $\%R^2$ metrics are computed for both training (not shown here) and testing sets. To keep the study brief but comprehensive, only the generalization (MSE, MAE and R^2 on the independent and unseen data, i.e., the error produced on the testing set) of the models are included in Table 1. Interestingly, the error profiles are sufficient to convey the performance and efficiency of the models. The extension of the results in terms of the simulation curves at two randomly selected biasing conditions are also given in Figs. 7–8.

As discussed earlier (see Section III-A), due to the nature of the backpropagation algorithms, they lack having inbuilt

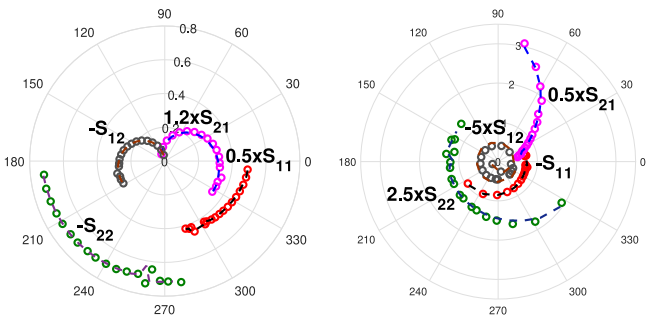

FIGURE 9. Measured (symbols) and predicted (dashed-lines) S-parameters at (a) $V_{GS} = -7$ V and $V_{DS} = 4$ V for four fingers GaN-on-SiC of 100 μm gate length each (b) $V_{GS} = -5$ V and $V_{DS} = 8$ V, for two fingers GaN-on-SiC of 200 μm gate length each (for GA-ANN models).

facility of finding the most appropriate initial weights and biases for the given problem. This problem may get further aggravated as it may force the gradients to acquire a very small value which in turn makes the training inefficient. This problem is more prominent when working with deeper networks, and obtaining unique solutions are quite challenging. The redressal of this issue led to the development of global optimization assisted ANN models [34]. The stringent requirement of obtaining the optimal weights and biases are satisfied by GA [11]. However, the parameters of the GA plays a pertinent role to get the optimal weights of the backpropagation algorithm. Even the hypeparameters of the GA are tunable and this further complicates the process. The solution adapted in this paper again is of trial-and-error (Section III-A). The ANN models are then developed, by embedding the optimal weights from GA by using the same ANN topologies, and then are trained and tested. The generalization capability of the models are given in Table 2. It can be inferred that the performance are improved after using the weights generated from the GA algorithm. The simulation and measurement s-parameters depicted in Figs. 9–10 at randomly selected bias conditions further proves the effectiveness of the models.

B. EVALUATION OF RSO-TUNED RANSAC MODEL

RANSAC models are developed in Python language making use of scikit-learn open source library. The training and

TABLE 3. Outcomes of the RANSAC models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	3.52e-2	2.81e-1	5.12e-1	4.18e-2	4.84e-2	5.19e-2	8.17e-2	1.89e-1
MAE	All bias	1.39e-1	2.81e-1	1.88e-1	1.75e-1	1.48e-1	1.69e-1	2.94e-1	2.57e-1
% R-Squared	All bias	44.51	8.73	8.52	38.14	2.9	45.44	8.86	9.35

TABLE 4. Outcomes of the SVR models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	1.78e-2	3.68e-2	4.15e-2	1.94e-2	1.98e-2	5.16e-2	6.67e-2	4.12e-2
MAE	All bias	9.41e-2	3.68e-1	1.54e-1	9.93e-2	8.84e-2	9.79e-2	1.78e-1	1.58e-1
% R-Squared	All bias	76.55	78.18	61.95	73.84	70.08	74.98	74.19	75.92

TABLE 5. Outcomes of the GPR models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	6.69e-3	8.15e-2	6.69e-4	5.48e-4	6.68e-4	8.93e-4	4.34e-3	3.55e-2
MAE	All bias	4.27e-2	1.91e-1	7.73e-3	9.02e-3	7.97e-3	1.29e-2	4.65e-2	8.12e-2
% R-Squared	All bias	72.15	83.44	98.25	98.15	97.11	97.10	97.33	96.69

testing sets are prepared according to the same procedure (see Section III). Models with preprocessing and post processing show improved performance [30]. Since the accuracy of the RANSAC models contingent on the design set without outliers, the extra step of outlier detection in the preprocessing stage is critical. The base estimator, maximum number of trials for random sample selection and loss functions are of primary importance. First, a working model is developed and default parameters of the algorithms are recorded. Then the range of the parameters are obtained using trial-and-error procedure. The parameters are tuned using RSO with 5-fold CV technique. It is important to note that RSO is simpler to implement in comparison to GA and therefore this has been used extensively in this paper. Subsequently, the number of iterations, a hyperparameter, to run the tuning process is determined from the error vs. iteration plot (not shown here). Based on the plot, it was found that running the algorithm for 250 iterations is sufficient for achieving the desired accuracy. Post tuning, the models are constructed using the optimal parameters. Then the training error (not shown) and testing error (Table 3) are calculated. The outcome of the tuning are the base estimator—multiple regressor for each S-parameter, the maximum number of iterations (between 30 to 150) for each S-parameter, and the optimal loss functions (i.e., the squared error loss function) for each S-parameter. Finally, the models are utilized to document the generalization capability in terms of MSE, MAE, $\%R^2$ metrics, shown in Table 3. From the table, it is obvious that the generalization capability of the RANSAC models are not good, especially for the phases of the S-parameters.

C. EVALUATION OF RSO-TUNED SVR MODEL

These SVR models are developed in MATLAB and the parameters are tuned using RSO with 5-fold CV technique. Once again the model development procedure outlined in Section III is used to develop the SVR models as well. First, a simple architecture is used to compute the training and testing errors that provide a foundation basis of the parameter's values. SVR employs kernel trick which facilitates in converting a non-linear low-dimensional problem to a linear high- dimensional problem. This trick makes use of simple dot product construct. In principle, SVR assists in solving complex and dynamic problems just by employing the kernel trick. There is a requirement to tune parameters namely Box constraint (C), Kernel scale, kernel function (KF), polynomial order, epsilon (ϵ) and presence and absence of the standardization of the data. Each parameter serves different purpose. For example, C acts as a trade-off between making the weights small and ensuring that each sample has functional margin of at least 1. In simple terms, it functions as a regularization parameter that penalizes any observation which goes beyond epsilon-insensitive loss region. It makes sure that the model neither overfits nor underfits for the given specification. For the decision boundary, the SVR concocts ϵ-insensitive region. This region stores the support vectors that in turn participates in the prediction process. So, tuning the right value of ϵ is paramount. Not all the KFs conduct in the same manner as they have their own hyperparameters which need to be tuned. During the extensive investigation, we have identified for the first time that the Gaussian kernel is the most efficient for each S-parameter of GaN HEMTs. Post tuning, the results obtained for each S-parameters are given below:

- mag. [S_{11}]— KF: Gaussian; C= 2.12; ϵ= 1.38e-4
- Phase [S_{11}]— KF: Gaussian; C= 2.64; ϵ= 1.45e-4
- mag. [S_{21}]— KF: Gaussian; C= 3.14; ϵ= 2.35e-5
- phase [S_{21}]— KF: Gaussian; C= 3.55; ϵ= 2.55e-5
- mag. [S_{12}]— KF: Gaussian; C= 1.89; ϵ= 1.81e-4
- phase [S_{12}]— KF: Gaussian; C= 1.93; ϵ= 1.93e-4
- mag. [S_{22}]— KF: Gaussian; C= 2.89; ϵ= 1.67e-4
- phase [S_{22}]— KF: Gaussian; C= 2.94; ϵ= 1.78e-4

Once the optimal values for the parameters are collected, the models are trained with complete training set. Both training

TABLE 6. Outcomes of the decision tree models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	6.01e-3	7.83e-3	5.56e-3	2.42e-3	4.99e-3	6.92e-3	5.48e-3	2.95e-3
MAE	All bias	4.42e-2	3.09e-2	2.49e-2	1.63e-2	4.35e-2	4.97e-2	3.72e-2	1.21e-2
% R-Squared	All bias	88.65	92.61	88.61	95.65	85.45	88.75	93.41	97.05

(not shown here) and testing errors are calculated. The generalization capability of the models are recorded and provided in Table 4. Even after multiple round of parameter tuning, the SVR could not produce results that could be comparable to ANN. The reasons behind this type of behavior is discussed in a later sub-section.

D. EVALUATION OF RSO-TUNED GPR MODEL

This model developed in MATLAB environment makes use of RSO as optimizer for tuning of the hyperparameters. It also uses expected-improvement-per-second-plus as acquisition function, and 5 fold CV for 30 iterations each. The number of iterations used for the tuning is smaller for GPR compared to other algorithms due to its computational inefficiency. It takes more time to tune and also requires a large memory space to store the overall model (see Table 16). The same procedure is applied to build the training and testing sets, preprocessing and post processing steps (see the methodology section). Kernel function (i.e., the covariance function) plays major role in the prediction process as GPR makes use of mean and covariance matrix for the prediction on any future input set. The tuning process starts once the central parameters are identified, and then the results are recorded. The tuned parameters for each S-parameter are given below, where abbreviations have the following meanings: FM (fit method), SD (subset of data points), BS (basis function), CS (constant function), EX (exponential), SX (squared exponential), RDE (relevance determination exponential), MK (matern kernel with parameter 5/2) and T (true). At last, exploiting the best parameters the models are trained and tested. Table 5 renders the generalization capability of the GPR model.

- mag. [S_{11}]— FM:SD; BS:CF; KF:EX; Standardize:T
- Phase [S_{11}]— FM:SD; BS:CF; KF:SX; Standardize:T
- mag. [S_{21}]— FM:SD; BS:CF; KF:RDE; Standardize:T
- phase [S_{21}]— FM:SD; BS:CF; KF:RDE; Standardize: T
- mag. [S_{12}]— FM:SD; BS:CF; KF: MK; Standardize: T
- phase [S_{12}]— FM:SD; BS:CF; KF:MK; Standardize: T
- mag. [S_{22}]— FM:SD; BS:CF; KF:MK; Standardize: T
- phase [S_{22}]— FM:SD; BS:CF; KF:MK; Standardize: T

E. EVALUATION OF RSO-TUNED DECISION-TREE MODEL

These models are developed in Python language by making use of scikit-learn open source library. As stated earlier, the parameters are tuned using RSO with 5-fold CV. The DTs are straightforward to build and uncomplicated to understand. However, the same supremacy can be a menace with regards to the generalization if they are not meticulously supervised as these algorithms are extremely prone to the overfitting. That means the parameter tuning is a central phase in order to build better SSMs using DTs. In general, DTs keep splitting the node until the nodes are pure. This can be overcome by setting the *maximum number of depth* to a fixed value. This enables the model to quit splitting no sooner the final value of maximum depth of tree is reached. Depending on the complexity of the tree, there is a requirement to split a node based on the number of samples present at the leaf. This is controlled and managed by "*minimum samples per leaf*". Splitting of the internal nodes are monitored by the parameter called *minimum samples to split* (MSS). The MSS generates a trade-off between the overfitting and underfitting case. Maximum number of features (MNF) can take on many values depending on the requirement of computational efficiency or overfitting. In scikit learn, the values it can take are *auto*, *sqrt*, and *log2*. In general, the MLF facilitates in determining the best split whenever a split takes place. Post tuning, the optimal values of the parameters recorded are given below. Here MDT (maximum depth of the tree), MNF (maximum number of features), MSL (minimum samples per leaf), MSS (minimum samples to split), CT (criterion), squared error (SE) and absolute error (AE).

- mag. [S_{11}]— MDT= 161; MNF = 5; MSL = 6; MSS =7; CT =SE
- Phase [S_{11}]— MDT= 186; MNF = 5; MSL = 11; MSS =13; CT =SE
- mag. [S_{21}]— MDT= 193; MNF = 5; MSL = 8; MSS =11; CT = SE
- phase [S_{21}]— MDT= 188; MNF = 5; MSL = 17; MSS =23; CT = SE
- mag. [S_{12}]— MDT= 194; MNF = 5; MSL = 13; MSS =19; CT = AE
- phase [S_{12}]— MDT= 198; MNF = 5; MSL = 15; MSS =22; CT = AE
- mag. [S_{22}]— MDT= 186; MNF = 5; MSL = 19; MSS =25; CT = SE
- phase [S_{22}]— MDT= 176; MNF = 5; MSL = 18; MSS =21; CT = SE

The models are built, trained, and tested using the procedure discussed in previous section. Finally, the generalization capability of the models are depicted in Table 6.

F. EVALUATION OF RSO-TUNED ENSEMBLE METHODS

All the ensemble models are developed in Python language making use of scikit-learn open source library and the parameters are tuned using RSO with 5-fold CV.

TABLE 7. Outcomes of the Bootstrap Aggregating models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	2.18e-3	5.89e-3	6.12e-3	5.55e-3	7.15e-4	3.22e-3	4.88e-3	7.42e-3
MAE	All bias	3.35e-2	6.68e-2	1.12e-2	2.26e-2	9.87e-3	9.21e-3	4.68e-2	7.54e-2
% R-Squared	All bias	97.62	96.08	96.43	97.02	97.07	97.23	97.35	97.49

TABLE 8. The hyperparameters for random forests and extremely randomized trees.

Parameters	No of estimators	Max depth of the tree	Min samples per leaf	Min samples to split	No of estimators	Max depth of the tree	Min samples per leaf	Min samples to split
	Random Forests				Extremely Randomized Trees			
Mag [S_{11}]	71	12	88	37	73	21	72	3
Phase [S_{11}]	85	14	52	3	99	23	74	13
Mag [S_{21}]	91	23	54	31	85	16	124	37
Phase [S_{21}]	83	21	67	32	87	18	129	39
Mag [S_{12}]	129	17	129	35	136	26	154	5
Phase [S_{12}]	783	18	156	51	189	34	159	18
Mag [S_{22}]	82	27	64	13	76	29	77	4
Phase [S_{22}]	96	13	69	19	72	28	84	12

TABLE 9. Outcomes of the random forests models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	1.81e-3	3.12e-3	3.25e-3	1.14e-3	7.78e-4	1.76e-3	2.21e-3	1.92e-3
MAE	All bias	1.72e-2	1.25e-2	1.31e-2	9.03e-3	7.51e-3	1.14e-2	1.93e-2	9.24e-3
% R-Squared	All bias	97.04	97.84	95.18	97.68	97.88	97.18	97.17	97.87

TABLE 10. Outcomes of the extremely randomized trees models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	1.82e-3	3.11e-3	2.55e-3	1.28e-3	5.25e-4	1.73e-4	1.92e-3	1.95e-3
MAE	All bias	1.84e-2	5.12e-2	2.11e-2	1.24e-2	8.92e-3	9.96e-3	1.57e-2	2.84e-2
% R-Squared	All bias	97.60	97.07	96.89	97.84	98.53	97.27	97.63	98.02

F.1. BOOTSTRAP AGGREGATING MODEL

Here the most critical parameters are the *base estimators* and the *number of the base estimators*. It aggregates the predictions of different base estimators either by voting or averaging in order to produce the final prediction. The base estimators could be chosen apart from the default DT. Another common base estimator is random forests (RF). Using these parameter the models are tuned for each S-parameter. Post tuning, the optimal values of the number of base estimators in this example is found to be 401 for mag. [S_{11}], 511 for Phase [S_{11}], 711 for [S_{21}], 741 for phase [S_{21}], 216 for mag. [S_{12}], 281 for phase [S_{12}], 551 for mag. [S_{22}] and 621 for phase [S_{22}]. Finally, the models are trained (not shown here) and tested, then its generalization capability are given in Table 7.

F.2. RANDOM FORESTS AND EXTREMELY RANDOMIZED TREES MODELS

In this paper, first the basic RF is implemented and then the ERTs are built. Both these models utilize the same technique to tune the parameters. Here, the hyperparameters that are tuned are *number of estimators*, *maximum depth of the trees*, *minimum samples per leaf*, and *minimum samples to split*. Each cover different operation. To control the number of decision tree estimators, we change the parameter number of estimators. Intuitively, it will make sense that the use of more number of trees will provide better results. However, in reality, the performance decreases after reaching a certain threshold. The maximum depth of the tree supervise the depth of each tree. Otherwise, if left unsupervised, it can lead to overfitting. Just like DT models, the minimum samples to split breakdown, or split the internal nodes if a certain number of samples, are present at a given moment. The minimum samples per leaf supervise the overfitting as it sets the minimum number of samples that has to be present at the leaf node in order for it to be considered as a split point. Each model is tuned for 250 iteration in this case and the optimized parameters are given in Table 8. Finally, performance are given in Tables 9 and 10 for RFs and ERTs respectively.

F.3. ADABOOST, GRADIENT TREE BOOSTING AND HISTOGRAM BASED GRADIENT BOOSTING MODELS

The idea and intuition behind the AdaBoost are very simple. It utilizes a regressor and trained on the original dataset alongside copies of the same regressor fitted at same dataset. However, to differentiate and make the prediction better, the weights are adjusted according to the error of the current prediction. Here, AdaBoost are trained with two base estimators, DT and RFs, and their performances are recorded

TABLE 11. Outcomes of the AdaBoost models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	1.82e-2	1.96e-2	2.63e-3	1.75e-2	1.61e-2	1.82e-2	3.95e-2	2.24e-2
MAE	All bias	1.32e-1	9.95e-2	1.24e-1	1.57e-1	9.41e-2	1.35e-1	1.81e-1	1.49e-1
% R-Squared	All bias	70.54	87.55	51.36	72.44	58.56	73.95	65.07	84.88

TABLE 12. Outcomes of the gradient tree boosting models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	6.18e-3	7.76e-3	9.95e-3	6.54e-3	5.24e-2	7.31e-3	9.97e-3	9.93e-3
MAE	All bias	5.97e-2	4.51e-2	6.37e-2	5.11e-2	4.54e-2	5.63e-2	8.55e-2	6.56e-2
% R-Squared	All bias	90.26	96.87	79.67	92.27	89.19	92.21	89.73	92.29

TABLE 13. Outcomes of the AdaBoost with RFs models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	1.11e-3	1.05e-3	3.81e-3	1.33e-3	3.14e-4	1.38e-3	1.81e-3	2.12e-3
MAE	All bias	2.97e-2	1.06e-2	5.76e-2	7.45e-2	8.75e-3	9.14e-3	4.07e-2	3.22e-2
% R-Squared	All bias	98.11	98.08	97.89	97.18	98.92	97.98	98.05	98.16

TABLE 14. Outcomes of the histogram based GT models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	2.23e-3	3.85e-3	4.07e-3	1.94e-3	9.25e-3	2.59e-3	2.89e-3	3.09e-3
MAE	All bias	2.33e-2	2.12e-2	2.07e-2	1.97e-2	5.36e-2	3.47e-2	3.29e-2	1.89e-2
% R-Squared	All bias	97.45	97.18	97.19	97.99	97.43	97.26	97.97	97.87

in Tables 11 and 13 respectively. This concept can be further studied as there could be numerous base estimators. The first hyperparameter, i.e., the *number of estimators*, are the number of weak learners to be used and the algorithm terminates after reaching the maximum value. Another vital parameter is the learning rate which employs the weights to the each regressor at each iteration. The contribution of each regressor is contingent on the learning rate, greater the learning rate, more the contribution and vice versa. From the above points it is incumbent to tune the parameters to get the optimal values for the hyperparameters. After tuning, the optimal number of estimators are found to be in the range of 100 to 250 for each S-parameter for both models. Similarly, the learning rate is in the range of 0.1 to 2 after tuning for all the models. Furthermore, post tuning, the hyperparameters of the gradient tree boosting and histogram based gradient tree boosting have somewhat similar values with a narrower margin. It is observed that the absolute loss and Huber loss give better performance. The tuned parameters for HGB is given below, where the abbreviations have the following meanings: MNI (maximum number of iterations), MNT (maximum number leaves for each tree), MNL (minimum number of leaves) and MDT (maximum depth of the tree). The generalization capability of GB and HGB are on the tuned parameters given in Tables 12 and 14.

- mag. [S_{11}]— MDT = 18; MNL = 70; MNT = 3; MDI = 85
- Phase [S_{11}]—MDT = 33; MNL = 186; MNT = 9; MDI = 194
- mag. [S_{21}]—MDT = 96; MNL = 1; MNT = 95; MDI = 198
- phase [S_{21}]—MDT = 190; MNL = 33; MNT = 44; MDI = 198
- mag. [S_{12}]—MDT = 37; MNL = 44; MNT = 87; MDI = 11
- phase [S_{12}]—MDT = 122; MNL = 14; MNT = 32; MDI = 71
- mag. [S_{22}]—MDT = 4; MNL = 14; MNT = 108; MDI = 95
- phase [S_{22}]—MDT = 171; MNL = 7; MNT = 69; MDI = 85

F.4. EXTREME GRADIENT BOOSTING MODEL

The XGBoost is one of the most popular algorithm for achieving accuracy and speed [50]. It has inbuilt flexibility for regularization that inherently assists the algorithm to avoid overfitting. To expedite the speed of the algorithm, it has inbuilt parallel processing unit. Maximum depth of the tree behaves similar to the GB. It oversees the overfitting of the models and decides on-the-fly. In the current example, post tuning, the values for the number of estimators and maximum depth are found to be 122, 198, 325, 348, 56, 74, 39, 41 and 18, 15, 10, 19, 14, 17, 11, 8 respectively for each S-parameter. The learning rate between 0.01 to 2 produced the most effective results. Finally, the generalization of the models on the tuned parameter is tabulated in Table 15. It is imperative to note that in this paper only a few hyperparameters are tuned and analyzed just to demonstrate the capability of this algorithm.

TABLE 15. Outcomes of the XGBoost models on the independent testing set (Generalization).

Metrics	Bias	Mag [S_{11}]	Phase [S_{11}]	Mag [S_{21}]	Phase [S_{21}]	Mag [S_{12}]	Phase [S_{12}]	Mag [S_{22}]	Phase [S_{22}]
MSE	All bias	3.21e-4	1.21e-3	1.34e-3	4.79e-4	1.33e-4	2.75e-4	1.27e-3	5.58e-3
MAE	All bias	3.87e-3	1.88e-2	2.07e-3	3.11e-3	1.33e-3	3.63e-2	9.39e-3	4.73e-2
% R-Squared	All bias	98.87	98.84	98.92	98.84	98.53	98.96	98.89	98.64

TABLE 16. Evaluation of all the proposed models for the modelling of GaN HEMTs devices.

Models	Parameters to compare						
	ADS compatibility	Computational efficiency	Average MSE	Training and simulation time (seconds)	Models' capacity	Parameters' tuning time (seconds)	Dataset compatibility
ANN	MATLAB - ADS Ptolemy	More Efficient	1.02e-3	6.125	Most	—	Recommended
GA-ANN	MATLAB - ADS Ptolemy	Efficient	4.96e-4	5.875	Most	7040	Recommended
RANSAC	ADS Datalink	Most efficient	1.55e-1	0.3595	Less	91.5	Not recommended
SVR	MATLAB - ADS Ptolemy	Less efficient	3.68e-2	4.75	Least	12250	Not recommended for larger non-linear dataset
GPR	MATLAB - ADS Ptolemy	Least efficient	3.18e-2	1804.85	Least	17408.75	Not recommended for larger non-linear dataset
DT	ADS Datalink	More Efficient	5.27e-3	0.2145	More	80.62	Recommended
BAM	ADS Datalink	Efficient	4.49e-3	24.77	More	1415.25	Recommended
RFs	ADS Datalink	Efficient	1.99e-3	38.75	More	2923.75	Recommended
ERTs	ADS Datalink	More efficient	1.66e-3	12.88	More	64	Recommended
AdaBoost	ADS Datalink	Efficient	1.92e-2	1.0625	Least	26.375	Not recommended
GB	ADS Datalink	More Efficient	7.86e-3	3.61	More	1025.62	Recommended
AdaBoost with RFs	ADS Datalink	Less efficient	1.61e-3	1211	More	14615	Recommended
Histogram based GB	ADS Datalink	Efficient	2.69e-3	20.75	Most	340.87	Recommended
XGBoost	ADS Datalink	Efficient	1.17e-3	2.2	Most	631.12	Recommended

V. RESULTS AND DISCUSSION

An assorted outcomes for the more commonly considered metrics namely ADS compatibility, computational efficiency, generalization capability, training and simulation time, models' capacity, parameters' tuning time and dataset compatibility are given in Table 16. Following discussions highlight the salient features of the models considered in this paper.

The first aspect in Table 16 compares the compatibility of all the developed models with a commercial Computer Aided Design (CAD) tool such as Advanced Design System (ADS). The ADS compatibility refers to the co-simulation of ML models. As elaborated in Section IV, the models in this paper are developed either using MATLAB or Python. The models in MATLAB renders an interface between MATLAB and ADS Ptolemy. It makes use of MATLAB blocks to prepare the input-output operations. Whereas, the ADS Datalink enables interconnection of the ADS with Python. Once the ML models are developed, the testing inputs can be given to trained models directly in Python. Thereafter, using the Datalink protocols, the output of the models can be successfully transferred into ADS environment. In this context, all the developed models are compatible with ADS and respective block diagrams in Figs. 11–12 clarify this. However, there are some challenges in forming the mathematical relationships of inputs, known parameters, and the outputs in several models such as GPR and SVR and therefore these suffer from some accuracy issues.

Computational efficiency in the Table 16 refers to the required memory, time taken, and the number of iterations to train the models. The investigations in this paper clearly conveys that the RANSAC is the most computationally efficient algorithm for model development. It requires least amount of memory in order for it to be trained and make prediction. It is the fastest among all other tested algorithms. It can also be seen that the ANN, DT, ERTs and XGBoost closely follow the RANSAC when the computational efficiency is considered. The SVR and AdaBoost with RFs are less efficient since they require more space to be stored and also their training and tuning time are high. The least efficient model is found to be the GPR models as it requires a large memory space to store the covariance matrix. In addition, the GPR

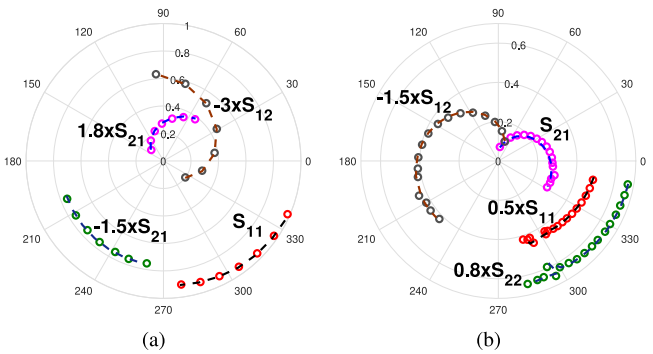

FIGURE 10. Measured (symbols) and predicted (dashed-lines) S-parameters at (a) $V_{GS} = -7$ V and $V_{DS} = 2$ V for four fingers GaN-on-SiC of 100 μm gate length each (b) $V_{GS} = -7$ V and $V_{DS} = 6$ V, for four fingers GaN-on-SiC of 100 μm gate length each (for GA-ANN models).

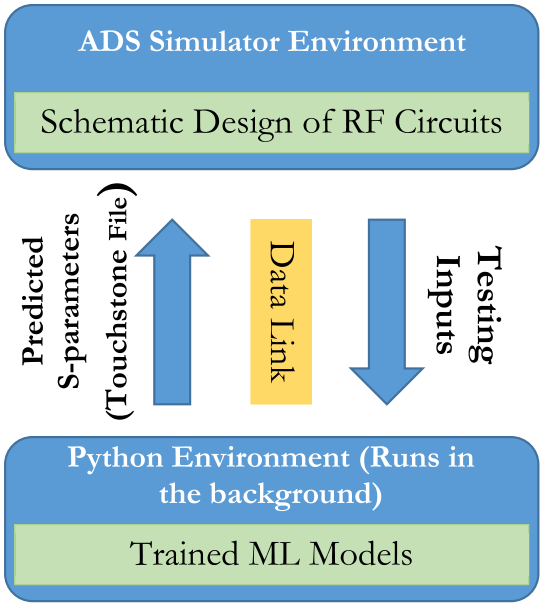

FIGURE 11. The block diagram explaining integration of Python based ML models with ADS simulator.

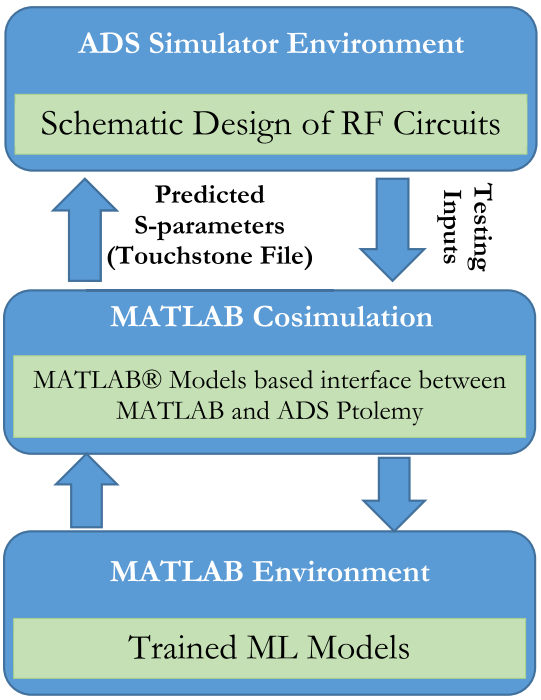

FIGURE 12. The block diagram explaining integration of MATLAB based ML models with ADS simulator.

is sparse and therefore they need entire training samples for prediction and this makes it computationally expensive.

Generalization capability denotes the model's performance over the unseen testing set. It is observed from the Tables 1 to 15 that the generalization capability of the GA assisted ANN based model produces the most accurate predictions on the testing set among all the models considered in this paper. This is owing to the fact that it captures the weights that orients towards the global minimum and also provides non-zero and non-vanishing gradient values. The performance of the ANN and XGBoost based models are comparable, with the former having a little edge due to the deep layer structure of the ANN model. Furthermore, the ANN models are also tuned more appropriately and hence provides better results. The XGBoost, however, shows excellent performance as it possess inbuilt hardware optimization efficient boosting mechanism of the weak learners, cross-validation capability, better handing of the overfitting, and regularization scenarios. Then the performance of the models decreases as we look from ERTs, RFs, Histogram based GB, GPR, BAM, DT, GB, AdaBoost, SVR, to RANSAC respectively. It can be safely stated that the tree-based models shows clear advantage in terms of generalization capability while demonstrating comparable performance to the ANN based models owing to the tabular nature of the data in these tree based algorithms. The GaN HEMTs devices have non-linear relationships and therefore the RANSAC based models do not produce good results as it makes use of linear decision boundary. The SVR and GPR take more time and are also computationally inefficient. It is therefore recommended to make use of linear kernels when applying SVR for larger dataset. However, linear kernel can not explain the intrinsic non-linear behavior possessed by GaN HEMTs devices. In the examples in this paper, we have included the dataset of two devices together and that may have resulted in a more sophisticated distribution within the data. Therefore, AdaBoost with DTs as their base estimator suffers heavily as it is highly prone to overfitting on unbalanced and noisy dataset. It is well-known that the SVR can not perform well in case of larger dataset and dataset with noise and sophisticated patterns [28], [35]. Since SVR's decision boundary becomes skewed on unbalanced dataset. Moreover, SVR exploits a subset of the data to predict since it does not take into account the observations which are within the ϵ-insensitive region. Similarly, larger dataset gives rise to the problems of data storage and processing in GPR. These lead to computational restriction and less accurate values of the predictions. In addition to this, computationally inefficiency directly affects the tuning of the hyperparameters and convergence of the SVR and GPR algorithms. In the Table 16, the Average MSE refers to the computation of average of MSE values of each model. It clearly summarises the overall generalization capability of models considered in this paper.

The training and simulation time for each model (averaged over all the S-parameters) in seconds) are given in Table 16. It is the time taken by an algorithm for it to be fully trained and then produce the prediction on the training and testing sets. The training and simulation time are computed individually for each S-parameter. Thereafter, we calculated the average time by adding the time taken for each S-parameter and then divide the total sum by the number of S-parameters and reported in the Table 16. It is also pertinent to mention that the training and simulation time are recorded, while training an algorithm with the optimal parameters and for

the run which has provided the best results. This process is repeated for each model. The device and windows specifications of the utilized computer to develop, train, tune and test all the models are as follows— Processor: Intel Core i7-8750H CPU @2.20 GHz 2.21 GHz, RAM: 16.0 GB, Edition: Windows 10 Enterprise and Version: 21H2. It is evident that the DT takes the least time as they employ greedy and parallel decisions in the fitting process. The RANSAC is the second fastest. It can also be seen that the XGBoost is one of the fastest and powerful model among all the considered models owing to the appropriate systematic split finding algorithm, parallelization, and the use of CPU cache memory for the calculation of gain on each split. The RFs and AdaBoost with RFs employ random forests which makes the predictions by calling multiple decision tree that increase the training and simulation time. Moreover, the time taken by tree like structures also depend on the maximum depth of trees. Better tuning of the hyperparameter improves the training and simulation time for all the tree based models tested in this paper. The SVR and GPR take more time since they require more memory to store the kernel matrix.

The time and complexity of the production of complete ML models are the *key indicators* for some specific applications of GaN HEMTs devices. That is why parameters' tuning time (averaged over all the S-parameters) in seconds and models' capacity are also given in the Table 16. Once again, the most expensive models in terms of the tuning time is GPR because it requires a large memory space to store the covariance matrix. The AdaBoost with RFs is the second costlier model followed by SVR, GA and RFs. The BAM and GB take the similar tuning time owing to the use of more or less same number of estimators or base estimators to employ the optimal models. The least expensive algorithms with regards to tuning time in the ascending order are AdaBoost, ERTs, DT, RANSAC, HGB, and XGBoost respectively. The ERTs are faster than RFs although they are from the same class of the algorithm and use the same procedure for the prediction. The RFs while choosing the split points looks for the optimal case whereas ERTs chooses randomly and thus achieve faster convergence.

Model's capacity refers to the parameters of the respective algorithms. Complexity of the model directly affects the performance of the model since a trade-off of all the parameters are required for the best prediction case. Parameter tuning becomes quite a daunting task when the model complexity becomes higher. The ANN, GA assisted ANN. XGBoost, and HGBs are the most complex models among the investigated models since they require tuning of more parameters for the effective prediction. The SVR, GPR, AdaBoost, and RANSAC have the least number of the parameters to tune. Table 16 gives the detailed information about the model's capacity. The last column in the Table 16 is dataset compatibility. This part of the table provides a recommendation to use different algorithms in specific application scenarios. It is clear that the RANSAC is not suitable for GaN HEMTs modelling as it relies on mostly linear decision boundaries. The SVR and GPR perform well in case of smaller dataset [30]. However, they are computationally highly expensive for larger sized data samples. The AdaBoost models can be trained with two base estimators namely the DT and RFs. It is observed (see Table 11) that AdaBoost with DT could not simulate well on the given problem. The ANN, GA-ANN, DT, BAM, RFS, ERTs, GB, AdaBoost with RFs, HGB, and XGBoost are recommended for the modelling of GaN HEMTs devices. However, the eventual use will depend on the application in hand as explained in the earlier paragraphs.

Lastly, following points outlines the summary and recommendation based on the results of this paper for the small signal modeling of GaN HEMTs:

- It is observed that GA initialized ANN produces the most accurate, robust, and efficient small signal model for GaN HEMTs devices. This is further validated by the simulation curves at distinct randomly selected biasing conditions. The applications where the required end results are the uniqueness of the solutions and accuracy then GA assisted ANN models should be preferred. However, it should be kept in mind that the tuning time of GA is higher and that eventually slows down the GA assisted ANN model development.
- The performance of ANN and XGBoost based ML models are quite similar. These models can be used for both smaller and larger datasets. They can be used for the applications where time is the most important parameter. However, both possess, a very diverse hyperparameters that are to be tuned in order to get good performance.
- RANSAC and AdaBoost are not appropriate for GaN HEMTs modelling
- SVR and GPR can be used for the smaller dataset where the need is to explain the behavior of GaN devices at lower frequencies.
- Tree based models have come out to be an excellent counter part of ANN based models due to nature of the dataset. These models can be utilized for GaN HEMTs modelling as they produce very good results for two distinct GaN devices considered in this paper. They can be used in the applications of GaN HEMTs devices where trade-off between time, complexity, and accuracy is desired. However, careful anlaysis of the parameters and trade-off between the parameters are necessary.

VI. CONCLUSION

In pursuit of determination of the most effective ML based small signal modeling, an exhaustive analysis and comparisons have been carried out. At first, various ML algorithms such as ANN, RANSAC, SVR, GPR, DT, GA based ANN, BAM, RFs, ERTs, AdaBoost, GB, HGB, and XGBoost are used to develop GaN HEMT modeling frameworks. Subsequently, the parameters of the models are tuned to get the best possibly performance from the models. The

models are then compared on grounds of generalization capability, ADS compatibility, computational efficiency, training and simulation time, models' capacity and parameters'tuning time. The generalization of the models are computed in terms of MSE, MAE, and R^2. It is identified that the GA initialized ANN based models produce the most accurate description of the small-signal behavior of GaN HEMTs devices. Furthermore, ANN and XGBoost based models' performance is found to be similar. The tree based models show excellent results. Finally, it is identified that the RANSAC, AdaBoost, SVR, and GPR are not suitable for the modeling of GaN devices. From the future perceptive, a more extensive research is required for GaN HEMTs modelling using the ensembling methods.

REFERENCES

[1] F. M. Ghannouchi and M. S. Hashmi, *Load-Pull Techniques with Applications to Power Amplifier Design*. Dordrecht, The Netherlands: Springer, 2012.
[2] S. Colangeli, A. Bentini, W. Ciccognani, E. Limiti, and A. Nanni, "GaN-based robust low-noise amplifiers," *IEEE Trans. Electron Devices*, vol. 60, no. 10, pp. 3238–3248, Oct. 2013.
[3] M. S. Cho, J. H. Seo, S. H. Lee, H. S. Jang, and I. M. Kang, "Fabrication of AlGaN/GaN fin-type HEMT using a novel T-gate process for improved radio-frequency performance," *IEEE Access*, vol. 8, pp. 139156–139160, 2020.
[4] M. Noweir, M. Helaoui, W. Tittel, and F. M. Ghannouchi, "Carrier aggregated radio-over-fiber downlink for achieving 2Gbps for 5G applications," *IEEE Access*, vol. 7, pp. 3136–3142, 2019.
[5] P. M. Tomé, F. M. Barradas, L. C. Nunes, J. L. Gomes, T. R. Cunha, and J. C. Pedro, "Characterization, modeling, and compensation of the dynamic self-biasing behavior of GaN HEMT-based power amplifiers," *IEEE Trans. Microw. Theory Techn.*, vol. 69, no. 1, pp. 529–540, Jan. 2021.
[6] F. Zeng et al., "A comprehensive review of recent progress on GaN high electron mobility transistors: Devices, fabrication and reliability," *Electronics*, vol. 7, no. 12, p. 377, Dec. 2018.
[7] F. M. Ghannouchi and M. S. Hashmi, "Load-pull techniques and their applications in power amplifiers design," in *Proc. IEEE Bipolar/BiCMOS Circuits Technol. Meeting*, Atlanta, GA, USA, Oct. 2011, pp. 133–137.
[8] T. Leng, K. Pan, X. Zhou, Y. Li, M. A. Abdalla, and Z. Hu, "Non-volatile RF reconfigurable antenna on flexible substrate for wireless IoT applications," *IEEE Access*, vol. 9, pp. 119395–119401, 2021.
[9] J. Kuzmík et al., "Self-heating in GaN transistors designed for high-power operation," *IEEE Trans. Electron Devices*, vol. 61, no. 10, pp. 3429–3434, Oct. 2014.
[10] K. R. Bagnall and E. N. Wang, "Theory of thermal time constants in GaN high-electron-mobility transistors," *IEEE Trans. Compon. Packag. Manuf. Techn.*, vol. 8, no. 4, pp. 606–620, Apr. 2018.
[11] A. Jarndal, S. Husain, M. Hashmi, and F. M. Ghannouchi, "Large-signal modeling of GaN HEMTs using hybrid GA-ANN, PSO-SVR, and GPR-based approaches," *IEEE J. Electron Devices Soc.*, vol. 9, pp. 195–208, 2021.
[12] R. Krishnamoorthy, N. Kumar, A. Grebennikov, and H. Ramiah, "A high-efficiency ultra-broadband mixed-mode GaN HEMT power amplifier," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 12, pp. 1929–1933, Dec. 2018.
[13] G. P. Gibiino, A. Santarelli, and F. Filicori, "A GaN HEMT global large-signal model including charge trapping for multibias operation," *IEEE Trans. Microw. Theory Techn.*, vol. 66, no. 11, pp. 4684–4697, Nov. 2018.
[14] R. G. Brady, C. H. Oxley, and T. J. Brazil, "An improved small-signal parameter-extraction algorithm for GaN HEMT devices," *IEEE Trans. Microw. Theory Techn.*, vol. 56, no. 7, pp. 1535–1544, Jul. 2008.
[15] W. Hu, H. Luo, X. Yan, and Y.-X. Guo, "An accurate neural network-based consistent gate charge model for GaN HEMTs by refining intrinsic capacitances," *IEEE Trans. Microw. Theory Techn.*, vol. 69, no. 7, pp. 3208–3218, Jul. 2021.
[16] J. R. Shealy, J. Wang, and R. Brown, "Methodology for small-signal model extraction of AlGaN HEMTs," *IEEE Trans. Electron Devices*, vol. 55, no. 7, pp. 1603–1613, Jul. 2008.
[17] F. Y. Huang, X. S. Tang, Z. N. Wei, L. H. Zhang, and N. Jiang, "An improved small-signal equivalent circuit for GaN high-electron mobility transistors," *IEEE Electron Device Lett.*, vol. 37, no. 11, pp. 1399–1402, Nov. 2016.
[18] G. Crupi, A. Caddemi, D. M. M.-P. Schreurs, and G. Dambrine, "The large world of FET small-signal equivalent circuits (invited paper)," *Int. J. RF Microw. Comput.-Aided Eng.*, vol. 26, no. 9, pp. 749–762, 2016.
[19] M. C. J. Weiser, J. Hückelheim, and I. Kallfass, "A novel approach for the modeling of the dynamic ON-state resistance of GaN-HEMTs," *IEEE Trans. Electron Devices*, vol. 68, no. 9, pp. 4302–4309, Sep. 2021.
[20] G. Crupi et al., "High-frequency extraction of the extrinsic capacitances for GaN HEMT technology," *IEEE Microw. Wireless Compon. Lett.*, vol. 21, no. 8, pp. 445–447, Aug. 2011.
[21] A. Jarndal, G. Crupi, M. A. Alim, V. Vadalà, A. Raffo, and G. Vannini, "Equivalent-circuit extraction for gallium nitride electron devices: Direct versus optimization-empowered approaches," *Int. J. Numer. Model.*, vol. 35, no. 5, p. e3008, 2022.
[22] A. Khusro, M. S. Hashmi, A. Q. Ansari, A. Mishra, and M. Tarique, "An accurate and simplified small signal parameter extraction method for GaN HEMT," *Int. J. Circuit Theory Appl.*, vol. 47, no. 6, pp. 941–953, 2019.
[23] Z. Marinković, G. Crupi, A. Caddemi, and V. Marković, "GaN HEMT small-signal modelling: Neural networks versus equivalent circuit," in *Proc. IEEE 30th Int. Conf. Microelectron.*, 2017, pp. 153–156.
[24] Z. Marinković, G. Crupi, A. Caddemi, V. Marković, and D. M. M.-P. Schreurs, "A review on the artificial neural network applications for small-signal modeling of microwave FETs," *Int. J. Numer. Model.*, vol. 33, no. 3, p. e2668, 2020.
[25] A. Khusro, S. Husain, M. S. Hashmi, and A. Q. Ansari, "Small signal behavioral modeling technique of GaN high electron mobility transistor using artificial neural network: An accurate, fast, and reliable approach," *Int. J. RF Microw. Comput.-Aided Eng.*, vol. 30, no. 4, 2020, Art. no. e22112.
[26] J. Cai, J. King, S. Chen, M. Wu, J. Su, and J. Wang, "Machine learning-based broadband GaN HEMT behavioral model applied to class-J power amplifier design," *Int. J. Microw. Wireless Technol.*, vol. 13, no. 5, pp. 415–423, 2021.
[27] A. Khusro, S. Husain, M. S. Hashmi, A. Q. Ansari, and S. Arzykulov, "A generic and efficient globalized kernel mapping-based small-signal behavioral modeling for GaN HEMT," *IEEE Access*, vol. 8, pp. 195046–195061, 2020.
[28] J. Cai, J. King, C. Yu, J. Liu, and L. Sun, "Support vector regression-based behavioral modeling technique for RF power transistors," *IEEE Microw. Wireless Compon. Lett.*, vol. 28, no. 5, pp. 428–430, May 2018.
[29] B. Scholkopf, *Support Vector Learning*. Munich, Germany: Oldenbourg-Verlag, 1997.
[30] S. Husain, A. Khusro, M. Hashmi, G. Nauryzbayev, and M. A. Chaudhary, "Demonstration of CAD deployability for GPR based small-signal modelling of GaN HEMT," in *Proc. IEEE Int. Symp. Circuits Syst.*, Daegu, South Korea, May 2021, pp. 1–5.
[31] S. Husain, A. Khusro, M. Hashmi, G. Nauryzbayev, and M. A. Chaudhary, "Gaussian process regression for small-signal modelling of GaN HEMTs," in *Proc. IEEE Int. Conf. Consum. Electron.*, Las Vegas, NV, USA, Jan. 2021, pp. 1–4.
[32] A. Khusro, M. S. Hashmi, A. Q. Ansari, and M. Auyenur, "A new and reliable decision tree based small-signal behavioral modeling of GaN HEMT," in *Proc. IEEE 62nd Int. Midwest Symp. Circuits Syst.*, 2019, pp. 303–306.
[33] S. Husain, K. Begaliyeva, A. Aitbayev, M. A. Chaudhary, and M. Hashmi, "Decision tree based small-signal modelling of GaN HEMT and CAD implementation," in *Proc. IEEE Int. Conf. Consum. Electron.*, Las Vegas, NV, USA, Jan. 2022, pp. 1–6.
[34] A. Jarndal, S. Husain, and M. Hashmi, "Genetic algorithm initialized artificial neural network based temperature dependent small-signal modeling technique for GaN high electron mobility transistors," *Int. J. RF Microw. Comput.-Aided Eng.*, vol. 31, no. 3, 2021, Art. no. e22542.

[35] A. Jarndal, S. Husain, and M. Hashmi, "On temperature-dependent small-signal modelling of GaN HEMTs using artificial neural networks and support vector regression," *IET Microw. Antennas Propag.*, vol. 15, no. 8, pp. 937–953, 2021.
[36] X. Long, Z. Jun, B. Zhang, D. Chen, and W. Liang, "A unified electrothermal behavior modeling method for both SiC MOSFET and GaN HEMT," *IEEE Trans. Ind. Electron.*, vol. 68, no. 10, pp. 9366–9375, Oct. 2021.
[37] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 281–305, 2012.
[38] S. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2007.
[39] S. Y. Lee et al., "An X-band GaN HEMT power amplifier design using an artificial neural network modeling technique," *IEEE Trans. Electron Devices*, vol. 48, no. 3, pp. 495–501, Mar. 2001.
[40] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
[41] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[42] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Stat. Comput.*, vol. 14, no. 3, pp. 199–222, 2004.
[43] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
[44] L. Breiman et al., *Classification and Regression Trees*. Monterey, CA, USA: Wadsworth Brooks, 1984.
[45] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[46] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[47] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.
[48] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class AdaBoost," *Stat. Interface*, vol. 2, no. 3, pp. 349–360, 2009.
[49] G. Ridgeway, "Generalized boosted models: A guide to the GBM package," *Update*, vol. 1, no. 1, p. 2007, 2007.
[50] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. KDD*, 2016, pp. 785–794.
[51] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proc. IJCNN Int. Joint Conf. Neural Netw.*, 1990, pp. 21–26.
[52] M. D. Pozar, *Microwave Engineering*. Hoboken, NJ, USA: Wiley, 2012.