http://pubs.acs.org/journal/acsodf

Article

Neuroevolution-Based Network Architecture Evolution in Semiconductor Manufacturing

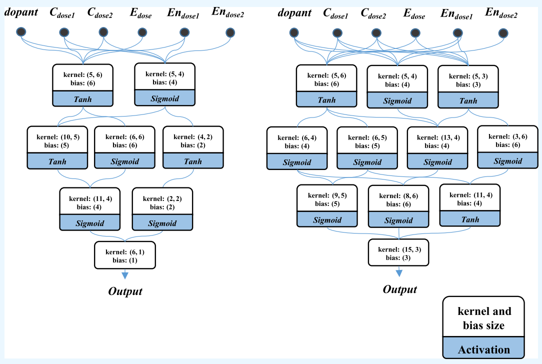
Yen-Wei Feng, Bing-Ru Jiang, and Albert Shihchun Lin*

Cite This: *ACS Omega* 2023, 8, 28877−28885

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Promoted model architectures or algorithms are crucial for intelligent manufacturing since developing them takes a lot of trial and error to embed the domain knowledge into the models correctly. Especially in semiconductor manufacturing, the whole processes depend on complicated physical equations and sophisticated fine-tuning. Therefore, we use a neuroevolution-based model to search the optimized architecture automatically. The collector current value at a particular bias of the silicon−germanium (SiGe) heterojunction bipolar transistor, generated by technology computer-aided design (TCAD), is used as the target dataset with six process parameters as the inputs. The processes include oxidation, dry and wet etching, implantation, annealing, diffusion, and chemical−mechanical polishing. Our work can build a suitable model network with a fast turnaround time, and practical physical constraints are fused in it without domain knowledge extraction. Take the case with 3840 data and one output as an instance. The mean square errors of the train set and validation set, as well as the mean absolute percentage error of the test set, are 1.317×10^{-6}, 7.215×10^{-7}, and 0.216 while using multilayer perceptron (MLP) and they are 3.285×10^{-7}, 1.661×10^{-7}, and 0.097 while using NE. The consequences show that the work in this vein is promising. According to the trend plot and results, the ability to extract physic is much better than the traditional (MLP) model.

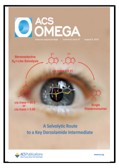

dopant
C_{dose1}
C_{dose2}
E_{dose}
En_{dose1}
En_{dose2}
kernel: (5, 6) bias: (6) Tanh
kernel: (5, 4) bias: (4) Sigmoid
kernel: (10, 5) bias: (5) Tanh
kernel: (6, 6) bias: (6) Sigmoid
kernel: (4, 2) bias: (2) Tanh
kernel: (11, 4) bias: (4) Sigmoid
kernel: (2, 2) bias: (2) Sigmoid
kernel: (6, 1) bias: (1)
Output
dopant
C_{dose1}
C_{dose2}
E_{dose}
En_{dose1}
En_{dose2}
kernel: (5, 6) bias: (6) Tanh
kernel: (5, 4) bias: (4) Sigmoid
kernel: (5, 3) bias: (3) Tanh
kernel: (6, 4) bias: (4) Sigmoid
kernel: (6, 5) bias: (5) Sigmoid
kernel: (13, 4) bias: (4) Sigmoid
kernel: (3, 6) bias: (6) Sigmoid
kernel: (9, 5) bias: (5) Sigmoid
kernel: (8, 6) bias: (6) Sigmoid
kernel: (11, 4) bias: (4) Tanh
kernel: (15, 3) bias: (3)
Output
kernel and bias size
Activation

■ INTRODUCTION

Many machine learning (ML) and deep learning technologies have been developed for intelligent manufacturing to deal with various tasks and apply them to process monitoring, process control, optimization, fault detection, and prediction.[1] Even in semiconductor manufacturing, sophisticated and intricate industries, ML methods can still be used for them. This includes using optimized model architectures based on traditional ML models[2] or innovative ones,[3] and improved optimization algorithms[4] in yield prediction and analysis, manufacturing process excursion detection, and manufacturing flow simplification.[5]

In model architectures, most current models are designed mainly by human experts, which is very time-consuming due to the need for sufficient domain knowledge and trial and error for suitable architectures. Therefore, the neural architecture search (NAS) or neuroevolution (NE) is a field attracting more attention as it can automatically find the optimized model for different cases by using search methods and fitting scores.[6] These methods are generally divided into several categories. First, the evolution-based methods, or neuroevolution (NE), where the genetic algorithm is the most regularly used technique in which a specific population of individuals is generated by algorithms, and a series of the process with selection, crossover, and mutation will be conducted after being evaluated and sorted according to their fitness score. Then, the new generation of individuals will develop in the direction of increasing the overall fitness from generation to generation until the termination condition is satisfied.[7,8] Second, the reinforcement learning (RL) method uses the accuracy of the child network defined by action space as a reward to set the training direction.[9] Third, the recently flourishing gradient optimization methods in which the model architecture composed of mixed operations are processed in a differentiable form and parameterized, and then, using the gradient descent method to evaluate the quality of the architecture.[10] Fourth, the surrogate model-based optimization can be regarded as a progressive model-constructing process. Its core concept is to search from a simple model, build a surrogate model of the objective function iteratively by recording past evaluations, gradually evolve to more complex architectures, and use it to predict the most promising architecture.[11−13] Bayesian Optimization (BO) is one of the most popular hyperparameter optimization methods, and many recent studies have attempted to apply them to NAS,[11,12] in which the validation results, in

Received: June 11, 2023
Accepted: July 17, 2023
Published: July 27, 2023



some cases,[11] are modeled as a Gaussian process (GP) guiding the search for an optimal architecture. Nevertheless, in GP-based BO methods, the inference time can cubically increase with the number of observations and cannot validly deal with variable-length problems.[14] Finally, other methods, including random search,[15] grid search,[16] simulated annealing,[17] etc., have been applied to execute NE/NAS. Although more and more applications of neural architecture search have been utilized in various problems,[18] in semiconductor fields, NAS studies have not been prevailing. Only a few studies using RL methods are located in the field of semiconductor manufacturing,[19] and almost none of them are based on NE or gradient optimization methods to optimize model architectures.

Achieving high accuracy needs a great deal of data which is sometimes tricky, costly, or impractical to attain.[20] Furthermore, data from science and engineering are apt to be dispersed and noisy because real-world experiments are expensive and subject to the environment and equipment. Hence, predictions may not be robust, owe interpretability, and even violate physical constraints, leading to generalization errors.[21] Integrating human knowledge into ML can help overcome these difficulties to some extent by significantly reducing data requirements, ameliorating reliability and robustness, and building interpretable ML systems.[20,22] Domain-knowledge-based ML is a practical approach employed in primarily scientific fields and can be generally classified into domain-knowledge-informed machine learning architecture, hybrid domain-knowledge-informed machine learning model, multifidelity framework, and Bayesian framework.[23] By embedding prior information into the neurons or layers, combining domain-knowledge-based models with ML models, or incorporating domain knowledge constraints into learning algorithms, it can enhance the information content of the available data and promote the learning algorithm to attain the correct solution. In this work, technology computer-aided design (TCAD) is used for simulating semiconductor processes under various parameters and producing datasets,[24] which offer sufficient, densely distributed data when experimental data measured from real devices are limited.

The MLP neural network is usually used as the basic structure in domain-knowledge-based ML. Still, its architecture must be adjusted first whenever it is applied to different tasks. Besides, accurately integrating prior domain knowledge into the architecture is also a conundrum. Thus, we take advantage of the automatic search ability of NAS to find out a particular network architecture as a type of domain knowledge constraint to help convergence. Previously, we have received some achievements in optimizing ML compact device models via NE algorithms.[25] In our work, the NE algorithm capable of dynamically tuning model architectures is utilized as a model optimization method to extract the physical characteristics of semiconductor devices manufactured under different process parameters. There have been very few studies in the field of semiconductors using NE with evolved topologies.[26] Therefore, we want to apply the method to clarify the effect of NE in semiconductor processes and device problems. Even in the entire field of chemistry and physics, NE-related works do not prevail, and after a careful literature review, we only found these studies[27]

■ METHODS

TCAD Simulations. The silicon−germanium (SiGe) heterojunction bipolar transistor (HBT) from Sentaurus TCAD examples is the device used in this work. The simulations are conducted using TCAD Sentaurus 2016 where SProcess[28] and SDevice[29] are both utilized to take into account the process effect on device characteristics. The advanced calibration for process simulation, the implantation models, and the diffusion models are considered in the process flow. First, the advanced calibration set of models and parameters guarantees accurate results for SiGe HBT fabrication during process simulation. Second, default analytic implantation tables are used for computing the profiles of the implanted dopants for each implantation step. The dual-Pearson distributions moment is extracted from Monte Carlo simulations with Crystal-TRIM. The point defect distribution, which is utilized to describe transient-enhanced diffusion (TED) phenomenon, is constructed via the "plus.one" model. The analytic Hobler model,[30] with a Gaussian primary function and an exponential tail, is used for calculating the damage to the crystal. As for diffusion models, the charged pair model is used. The dopant-point defect pairs and unpaired point defects are treated as movable species, and the substitutional dopants are treated as immobile species. The coupled system of three continuity equations, i.e., interstitials, vacancies, and unpaired dopants, is thus formed. Moreover, germanium diffusion and GeB cluster formation[31] add two more equations to the system. The whole process flow contains ten main steps, including isolation trench definition, subcollector implantation, subcollector activation, polysilicon base−contact deposition, the definition of base opening, SiGe base deposition, collector implantation, nitride spacer definition, final anneal, and metallization. Before the first step, the initial mesh set-up of the substrate is defined. In the first step, the isolation trench is defined and etched by the first mask, and then, the oxide is deposited after the photoresist is stripped. In the second and third steps, for the subcollector process, pre-implantation and primary implantation are processed with the second mask before the second photoresist stripping. Afterward, the device is annealed for several seconds by rapid thermal annealing (RTA), and the oxide is etched. Subsequently, the base area of HBT is defined, which contains polysilicon doped by boron with a doping concentration of 1×10^{20} cm^{-2}. Then, the subsequent structures of the device are specified through the third and fourth masks. Oxide and nitride are deposited after polysilicon etching, exposure and development with the third mask. Afterward, the silicon nitride, silicon oxide, and polysilicon are etched one by one before photoresist stripping. Then, another run of nitride deposition and etching of nitride and oxide is conducted with the fourth mask. The fourth step to the sixth step is to cope with the strained SiGe base, in which a 0.2 μm SiGe layer is deposited first, and the layer is etched back by chemical−mechanical polishing to a final thickness of 0.1 μm. In collector implantation, the oxide layer is deposited after the nitride stripping, and then, this region is doped with an n-type dopant before stripping the oxide. In the next step, the nitride spacer is defined by the nitride deposition diffused under 900 °C and nitride etching. Later, the emitter doped with an n-type dopant is defined, and the fifth mask is used in polysilicon etching. In the final annealing, the device is annealed to reconstruct the bonding between the dopants and silicon and activate the dopants after the oxide deposition. Finally, the metallization is performed to form the contacts of the device by the sixth mask. In this section, six variables are taken into account, including the dopant types and the implant dose of the emitter implant, collector pre-implant, and collector main implant. In addition, the implant energy of the collector implant

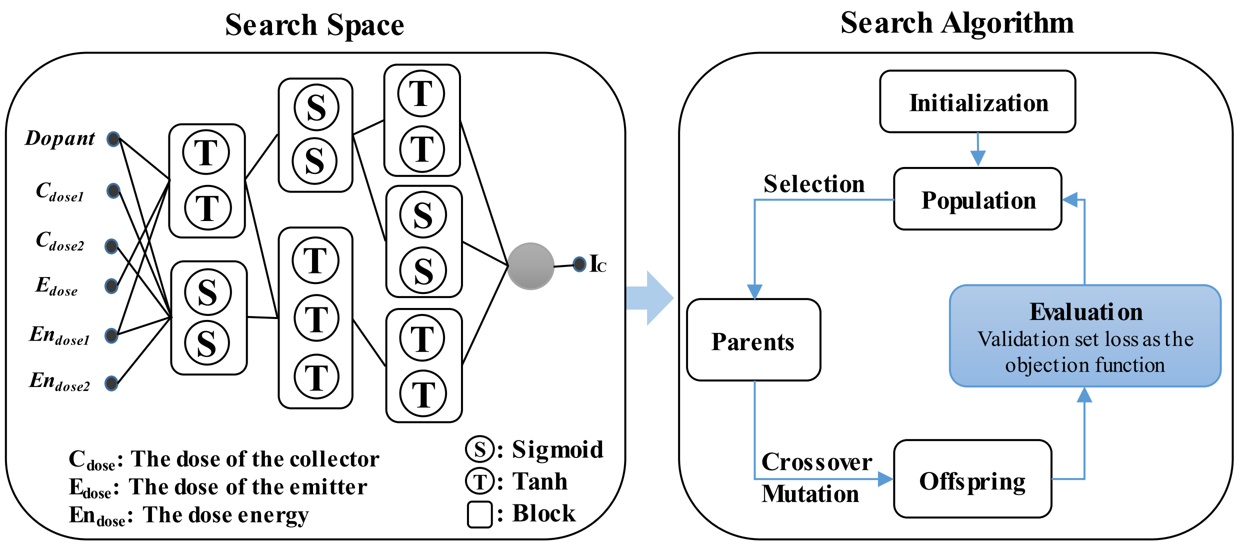


Figure 1. Workflow of Neural Architecture.

is also changed. They are varied uniformly within specific ranges, which are two kinds of ions, arsenic, and phosphorus, doses of the subcollector for pre-implant, 1×10^{15} and 5×10^{15} cm^{-2}, doses of the collector for main implant, 5×10^{15} to 1×10^{16} cm^{-2} with step 1×10^{15} cm^{-2}, dose energy of the collector for pre-implant, 40 to 400 keV with step 40 keV, dose energy of the collector for main implant, 30 to 100 keV with step 10 keV, as well as dose energy for the emitter, 1×10^{20} and 5×10^{20} cm^{-2}. The setting ensures that $N_E > N_B > N_C$, where N is doping concentration and E, B, and C are symbols for emitter, base, and collector, respectively. These variables are used as inputs in ML.

In device simulations, necessary models are chosen to attain a multidisciplinary simulation. For instance, doping dependence and high field saturation models are used for carrier mobility description. Auger recombination, Schottky−Reed−Hall recombination (SRH), and Avalanche breakdown models are used to describe recombination. The hydrodynamic model is used to describe the transportation of carriers in semiconductor devices. The effective intrinsic density model is used to describe band gap narrowing. Poisson equations with the electron and the hole continuity equations are also solved. Then, the base of HBT is changed to the current mode and set to three values from 2×10^{-6} to 4×10^{-6} A. The collector voltage (V_{ce}) is ramped in quasistationary simulations from 0 to 0.5 V. At last, the collector current (I_c) values at a certain V_{ce} and each base current are extracted with Matlab and used as outputs in ML.

Neuroevolution Algorithm. In this study, python3.8.15,[32] Tensorflow2.7.0,[33] Numpy1.21.6,[34] Pandas1.2.4,[35] Scikit-Learn 0.24.1,[36] and Matlab R2021a[37] are used. The MLP model is used as a baseline. The dataset size in this work is less than 5000.

Previously, we used a NE-based model with genetic algorithms (GA) in transistor compact device modeling and achieved satisfactory results.[25] In this work, the NE method is applied to searching the neural network architecture shown in Figure 1, which can find the optimized model to extract domain knowledge in semiconductor manufacturing. Initially, the input parameters are denoted as dopants, Cdose1, Cdose2, Endose1, Endose2, and Edose for the abovementioned parameters. TCAD simulation data are randomly divided into a training set and a test set for model training and testing. Data are preprocessed before training, which can help converge faster within gradient descent. The values of the dose of the collector, the dose of the emitter, and the collector current are taken logarithm of, and then, the dataset is normalized. The GA are utilized as the NAS method to modify the network architecture, and the genomes can be effectively assembled with the encoding strategy to facilitate subsequent operations such as crossover and mutation in the entire network structure. In the encoding strategy, the blocks are considered as units, and each layer is composed of blocks with different numbers of neurons and activation functions, i.e., hyperbolic tangent (tan*h*) and sigmoid. The connections between each layer are chosen by uniform selection, and there are some restrictions on connections between the blocks in each layer. First, the first hidden layer should take three to six input parameters as connections. Second, each layer can only be selected and connected by the previous layer. Finally, the output layer should select at least half of the blocks from the previous layer. In the process, layer numbers, block numbers in each layer, neurons, and activation functions are all used in the search space to find the optimal network architecture. The MLP model training and validation are performed with the number of hidden layers set at five and the neurons of each layer set between five and ten. The MLP baseline set in this manner has similar parameter numbers in reference to NE-based models. The early stopping method are also applied to MLP and NE model with patience = 500. As for the model with GA, the parameters of the search space are mainly decided to be <500 within the setting range of two to five hidden layers, one to four blocks, and one to six neurons to carry out the network architecture search. Every population has N individuals. At first, the initial generation should randomly generate the genomes of the individuals based on the setting. Then, the genomes of each individual will convert to the neuroevolution-based model whose networks will be trained through an adaptive moment estimation (Adam) optimizer and be evaluated by the validation set simultaneously. After that, the weights will be saved according to the loss from the validation set as the target functions for GA. Since the validation set is from the training set, it can effectively assess the ability of the networks to the untrained data and can avoid overfitting. The random selections for individuals chosen from a population that can maintain genetic diversity will select a pair of parents, depending on the fitness scores, to conduct crossover and mutation to reproduce their offspring. The crossover operation can swap the

Table 1. MLP and NE Results on SiGe HBT Semiconductor Processing Datasets with 6 Inputs and 1 Outputs Networks Used in This Case, and Training Epochs Are 20,000 with the Patience 500 of Early Stop Setting and Restoration of the Best Weights

model architecture	dataset size	train-test split	epochs	train set MSE	train set MAPE (%)	validation set MSE	validation set MAPE (%)	test set MSE	test set MAPE (%)	numbers of parameters
multilayer perceptron	1920	0.2	2373	1.095×10^{-4}	1.515	1.109×10^{-4}	1.511	1.274×10^{-4}	1.692	353
		0.5	7066	1.160×10^{-6}	0.166	8.905×10^{-7}	0.173	9.880×10^{-7}	0.183	281
		0.8	6136	6.305×10^{-7}	0.129	4.746×10^{-7}	0.125	5.985×10^{-7}	0.139	217
	2880	0.2	6220	1.001×10^{-5}	0.474	1.088×10^{-5}	0.542	2.374×10^{-5}	0.709	217
		0.5	8837	5.114×10^{-7}	0.103	5.774×10^{-7}	0.117	4.473×10^{-7}	0.112	281
		0.8	4208	4.695×10^{-6}	0.292	2.942×10^{-6}	0.300	2.862×10^{-6}	0.298	217
	3840	0.2	8144	2.383×10^{-6}	0.350	2.851×10^{-6}	0.407	4.139×10^{-6}	0.459	353
		0.5	5353	5.047×10^{-7}	0.146	3.537×10^{-7}	0.157	3.490×10^{-7}	0.152	353
		0.8	8132	1.317×10^{-6}	0.210	7.215×10^{-7}	0.222	6.861×10^{-7}	0.216	217
neuroevolution	1920	0.2	7554	1.480×10^{-7}	0.070	2.007×10^{-7}	0.077	2.589×10^{-7}	0.086	346
		0.5	9200	1.283×10^{-7}	0.045	6.031×10^{-8}	0.041	8.690×10^{-8}	0.049	258
		0.8	9885	1.103×10^{-7}	0.045	6.578×10^{-8}	0.046	7.711×10^{-8}	0.051	228
	2880	0.2	11,054	4.996×10^{-7}	0.088	4.172×10^{-7}	0.095	7.624×10^{-7}	0.121	203
		0.5	9155	2.722×10^{-7}	0.062	2.231×10^{-7}	0.067	2.026×10^{-7}	0.069	258
		0.8	6935	3.016×10^{-7}	0.069	1.561×10^{-7}	0.071	1.695×10^{-7}	0.069	228
	3840	0.2	7043	3.303×10^{-7}	0.125	2.937×10^{-7}	0.142	3.182×10^{-7}	0.142	343
		0.5	4940	2.803×10^{-7}	0.104	1.739×10^{-7}	0.107	1.872×10^{-7}	0.107	346
		0.8	6904	3.285×10^{-7}	0.099	1.661×10^{-7}	0.097	1.582×10^{-7}	0.097	228

Table 2. MLP and NE Results on SiGe HBT Semiconductor Processing Datasets with 6 Inputs and 3 Outputs Networks Used in This Case, and Training Epochs Are 20,000 with the Patience 500 of Early Stop Setting and Restoration of the Best Weights

model architecture	dataset size	train-test split	epochs	train set MSE	train set MAPE(%)	validation set MSE	validation set MAPE(%)	test set MSE	test set MAPE(%)	numbers of parameters
MLP	1920	0.2	8023	5.607×10^{-6}	0.403	6.355×10^{-6}	0.444	7.465×10^{-6}	0.476	297
		0.5	3716	6.984×10^{-6}	0.456	3.770×10^{-6}	0.370	7.007×10^{-6}	0.455	297
		0.8	5383	3.058×10^{-7}	0.092	2.517×10^{-7}	0.096	2.569×10^{-7}	0.096	453
	2880	0.2	10,843	6.271×10^{-6}	0.366	7.191×10^{-6}	0.439	7.515×10^{-6}	0.453	231
		0.5	5621	9.154×10^{-7}	0.152	8.010×10^{-7}	0.158	7.904×10^{-7}	0.164	371
		0.8	7086	9.709×10^{-7}	0.166	8.011×10^{-7}	0.172	7.960×10^{-7}	0.171	297
	3840	0.2	7283	2.133×10^{-6}	0.327	4.059×10^{-6}	0.475	4.616×10^{-6}	0.433	543
		0.5	3745	2.486×10^{-6}	0.378	2.443×10^{-6}	0.397	2.338×10^{-6}	0.395	453
		0.8	6480	8.150×10^{-7}	0.208	6.943×10^{-7}	0.211	6.003×10^{-7}	0.201	453
NE	1920	0.2	20,000	2.564×10^{-7}	0.079	2.487×10^{-7}	0.089	2.798×10^{-7}	0.097	272
		0.5	16,594	1.283×10^{-7}	0.059	1.129×10^{-7}	0.066	1.128×10^{-7}	0.065	272
		0.8	5078	2.505×10^{-7}	0.074	1.564×10^{-7}	0.075	1.593×10^{-7}	0.076	488
	2880	0.2	19,633	4.930×10^{-7}	0.123	4.917×10^{-7}	0.130	7.505×10^{-7}	0.151	242
		0.5	12,477	5.497×10^{-7}	0.104	3.417×10^{-7}	0.107	3.383×10^{-7}	0.107	365
		0.8	5113	3.613×10^{-7}	0.089	2.737×10^{-7}	0.094	2.254×10^{-7}	0.087	272
	3840	0.2	13,761	5.657×10^{-7}	0.161	5.273×10^{-7}	0.189	5.743×10^{-7}	0.190	499
		0.5	15,175	5.445×10^{-7}	0.172	3.803×10^{-7}	0.170	3.801×10^{-7}	0.169	468
		0.8	11,437	2.455×10^{-7}	0.112	1.969×10^{-7}	0.116	1.716×10^{-7}	0.111	421

tail of the genome of the parents, and the mutation is used to fix the connections loosened because of the crossover operation. The process will be repeated until the number of offspring attain N individuals. Then, a group coupled with parents and offspring will be sorted by the fitness score and selected as a new population. During the continuous evolving process, the population will gradually converge to a particular network architecture, which is the optimal model for the sequence of semiconductor processes.

The computer with CentOSn 5.7, Intel Core i5−3470 CPU @ 3.20GHz, and 20GB RAM is used for TCAD simulations, while the computer with Windows 10 64-bit, Intel Core i7−12,700 @2.1GHz, and 16GB RAM is used for NAS.

■ RESULT AND DISCUSSION

Table 1 shows the MLP and NE-based models trained under six inputs with one I_c value as an output. In each dataset size, the train set is divided into different proportions, and the rest is the test set. During training, 20% of the train set is assigned to the validation set, and training is conducted for 20,000 epochs with the patience 500 of early stop setting. The best case of the MLP models selected among multiple architectures is used to compete with the optimized model based on NE. It should be emphasized that both MLP and NE models are saved at the lowest validation loss during training, i.e., restoring the best weights. From the performance on either the training or the test set, NE is better than MLP. It is observed that the optimized model architecture of the NE-based model can attain better results with fewer iterations when using gradient descent

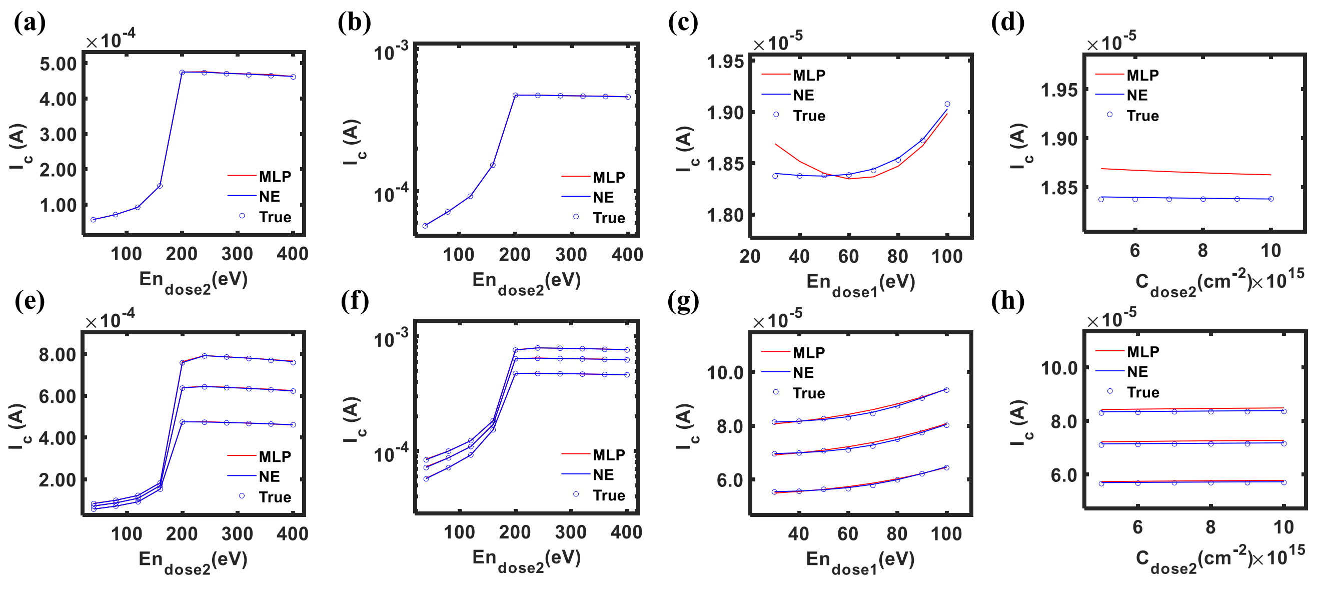


Figure 2. Trend charts by the prediction of MLP and NE with dataset size = 3840 and train-test split = 0.8. (a) $I_c - En_{dose2}$ in the linear scale with one output, (b) $I_c - En_{dose2}$ in log scale with one output, (c) $I_c - En_{dose1}$ with one output, (d) $I_c - C_{dose2}$ with one output, (e) $I_c - En_{dose2}$ in the linear scale with three outputs, (f) $I_c - En_{dose2}$ in the log scale with three outputs, (g) $I_c - En_{dose1}$ with three outputs, and (h) $I_c - C_{dose2}$ with three outputs.

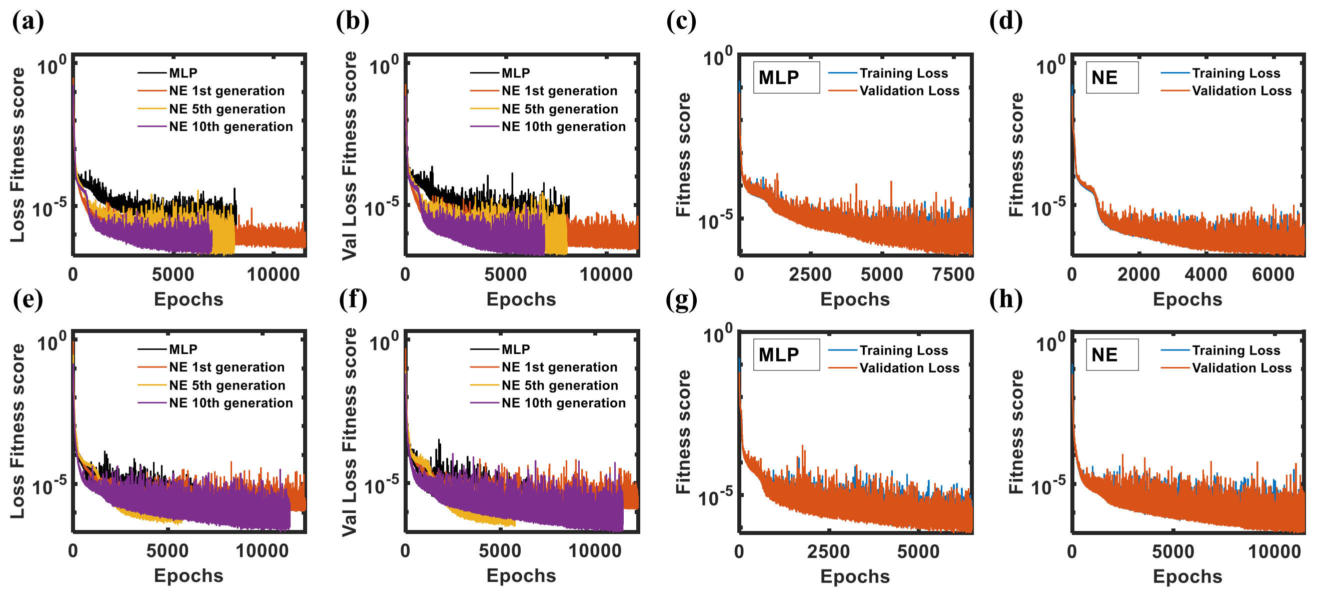


Figure 3. MLP and NE models' fitness score with dataset size = 3840, train-test split = 0.8. (a) MLP and NE model's loss in the train set with one output. (b) MLP and NE model's loss in the validation set with one output. (c) MLP model's training loss and validation loss comparison with one output. (d) NE model's training loss and validation loss comparison with one output. (e) MLP and NE model's loss in the train set with three outputs. (f) MLP and NE model's loss in the validation set with three outputs. (g) MLP model's training loss and validation loss comparison with three outputs. (h) NE model's training loss and validation loss comparison with three outputs.

regardless of the dataset sizes or train-test splits. This phenomenon also indicates the effect of data efficiency. The dataset sizes and train-test splits in Table 2 are the same as the conditions in Table 1, but the only difference is that the three I_c values at different biases are used as the outputs in MLP and NE-based models. The purpose of training under six inputs and three outputs is to increase the complexity of the problems to show the superior fitting and domain-knowledge extraction capability of NE. The results externalize that NE can still outperform MLP in this case.

As shown in Figure 2, the domain-knowledge trends are plotted to see the effect of NE for the different cases in Tables 1 and 2. Figure 2a−d shows predictions under 3840 data, train-test split = 0.8, and one output. Figure 2a,b shows the linear graphs and logarithm graphs of I_c to En_{dose2}, while Figure 2c,d shows graphs of I_c to En_{dose1} and C_{dose2}, respectively. As shown in Figure 2a,b, the predictions of MLP are roughly the same as those of NE. Nevertheless, in Figure 2c,d, due to the slight change in collector current, it is evident that NE is better than MLP at predicting the domain-knowledge trends, showing the fitting capability and domain-knowledge extraction of NE. Figure 2e, f shows images similar to those in Figure 2a−d, which are only changed to three outputs. Since the simultaneous prediction of three outputs enhances the complexity of the model, the poor predictions of the curves in MLP are observed in Figure 2e−f. In

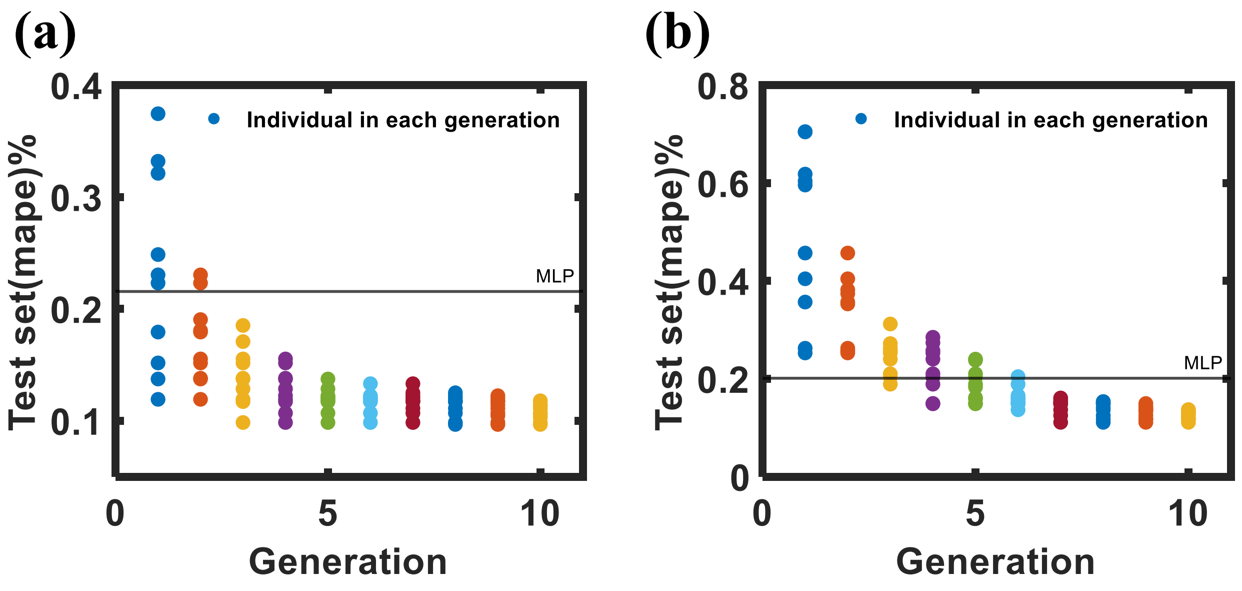


Figure 4. (a) MLP and NE models' performance in the test set with six inputs and one output. Dataset size = 3840 and train-test split = 0.8. (b) MLP and NE models' performance in the test set with six inputs and three outputs.

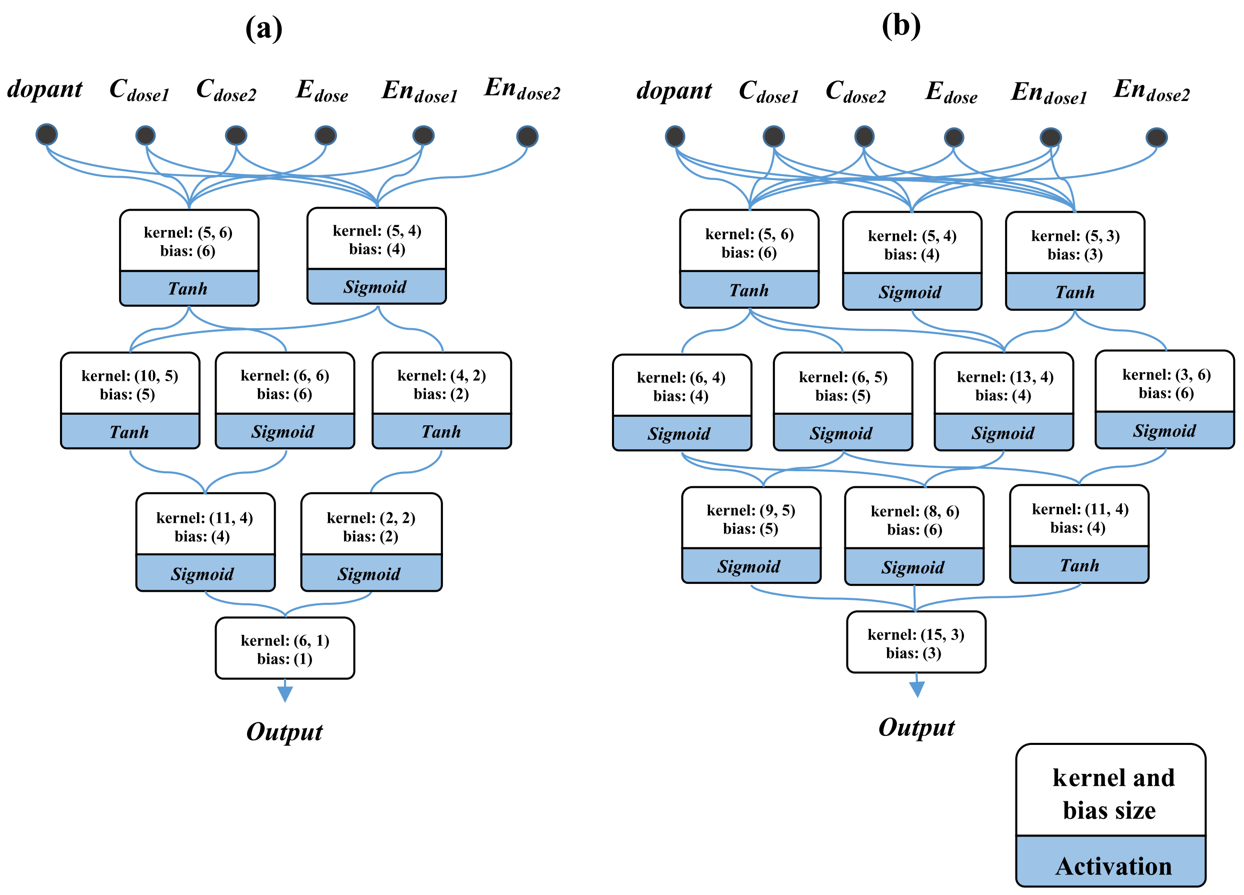

(a)
dopant
C_{dose1}
C_{dose2}
E_{dose}
En_{dose1}
En_{dose2}
kernel: (5, 6) bias: (6) Tanh
kernel: (5, 4) bias: (4) Sigmoid
kernel: (10, 5) bias: (5) Tanh
kernel: (6, 6) bias: (6) Sigmoid
kernel: (4, 2) bias: (2) Tanh
kernel: (11, 4) bias: (4) Sigmoid
kernel: (2, 2) bias: (2) Sigmoid
kernel: (6, 1) bias: (1)
Output
(b)
dopant
C_{dose1}
C_{dose2}
E_{dose}
En_{dose1}
En_{dose2}
kernel: (5, 6) bias: (6) Tanh
kernel: (5, 4) bias: (4) Sigmoid
kernel: (5, 3) bias: (3) Tanh
kernel: (6, 4) bias: (4) Sigmoid
kernel: (6, 5) bias: (5) Sigmoid
kernel: (13, 4) bias: (4) Sigmoid
kernel: (3, 6) bias: (6) Sigmoid
kernel: (9, 5) bias: (5) Sigmoid
kernel: (8, 6) bias: (6) Sigmoid
kernel: (11, 4) bias: (4) Tanh
kernel: (15, 3) bias: (3)
Output
kernel and bias size
Activation

Figure 5. (a) Optimized NE model by GA with six inputs and one output. Dataset size = 3840 and train-test split = 0.8. (b) Optimized NE model by GA with six inputs and three output.

this case, it can also be observed that NE's fitting capability and domain-knowledge extraction is better, as shown in Figure 2g,h.

Figure 3a−d are the one-output cases, and Figure 3e−h are for three outputs. In (a), (b), (e), and (f), train loss and validation loss of the best individual in a generation of NE are selected. They indicate that after the evolution of NE, the convergence speed of the architecture training process increases in both the train set and validation, stably surpassing the performance of MLP. In (c), (d), (g), and (h) the training loss and the validation loss of the model with one output and three outputs during training are shown. (c) and (g) show both for MLPs, while (d) and (h) are all models after NE optimization. The graphs reveal that the optimized NE model has a fast convergence speed and less fluctuation.

Finally, Figure 4a,b shows the performance of NE and MLP models in 1 output and three output cases, respectively. The graphs record the accuracy of each generation of NE and the best hand-tuned MLP for the predictions from the test set, in which the mean absolute percentage error (mape) is used as an evaluation. At the beginning of the process, the mape scores of each individual of NE are relatively scattered. However, the individual of each generation in the search space will be stably evolved and converged to a specific architecture that is good at predicting the domain knowledge trend of TCAD simulations, and then, the optimal model is obtained, as shown in Figure 5a,b.

In smart manufacturing, NAS is suitable for solving the issue that the development time of domain-knowledge-based model construction is time-consuming. NE methods are used as a strategy to search the entire search space, such as neural networks through augmenting topologies (NEAT).[7] By the genetic algorithm, NEAT evolves architecture and weights incrementally through adding neurons and removing connections, which can continuously explore search space. However, for the complex nonlinearity in the dataset, NEAT only uses the genetic algorithm to solve it, and it may take a long time to approach excellent performance compared to gradient descent. In our work, Adam is used to train the network, and the genetic algorithm is used to evolve the architecture and weights. Selection operation randomly chooses parents to reproduce, which maintains the genetic diversity of networks. Parts of the parents' characteristics are kept under various operations, such as crossover and mutation, and the architectures are robustly evolved. In the way of the gradient descent, the nonlinearity issue can be solved, and the genetic algorithm can find the optimized model for domain-knowledge extraction.

In the field of using domain knowledge in ML, Li et al. eliminate the nonphysical behavior produced from MLP model predictions by embedding domain knowledge into layers of the model. The domain knowledge is to use the tan*h* function to fit the linear region of $I_D - V_{DS}$ at small V_{DS} and the saturation region at large V_{DS} in thin-TFET devices. Besides, sigmoid functions are used to describe the $I_D - V_{TG}$ curve in the subthreshold region that turns on exponentially and then becomes polynomial in the on region. Using these two activation functions as the domain-knowledge constraints can solve the counter-theory prediction caused by the lack of devices' domain knowledge in a specific part of the MLP model.[38] Kao et al. also proposed a hybrid physics-based BSIM model and ML model architecture, where the output of the BSIM is the current value and the output of the ML model is a bias-dependent correction function $\varepsilon(V_{GS}, V_{GD})$ for the nonidealities not included in the BSIM model. The output of the BSIM and ML models are multiplied to obtain the final current value. This method can achieve satisfactory generalization and Gummel symmetry of devices in most device operation regions.[39] Although the above two methods can successfully reach their aim, they still need human tuning in ML model architectures to find a better one. In our approach, our model uses the GA-based NAS method to generate the model architecture and activation function selection, which is a fully automated process. As long as the search rules of the algorithm are given, the architecture search can be automatically tuned without trial and error by human hands. Compared with the MLP model, the NE model shows better generalization and is closer to the actual value in the result prediction.

The nonlinearity and high complexity of the dataset is a significant problem in many fields. The MLP model makes use of an activation function to solve this problem. The domain knowledge itself can be complicated and thus results in high data nonlinearity. Therefore, MLP that only uses a single activation function often cannot effectively solve it. Although using an MLP model with more parameters can help fit a dataset with complex changes, it is also more likely to cause the model to overfit the training set. Eventually, the model's prediction accuracy on the test set will decline. Consequently, the GA-based NE method employed in this work is to adjust the model architecture, which is tantamount to integrating domain knowledge constraints into the framework after modifications. Combining the connections with two activation functions between blocks here provides another assistance to the architecture. It helps nonlinearity fitting since the connections are selected by GA with a uniform random selection scheme to ensure the search space's diversity. Parents' strengths are preserved for the next generation through crossover and mutation to evolve the network architectures robustly. The whole process here is to limit the counter-theory behavior that the MLP model is prone to predict because of insufficient data information. Thus, our work can retain generalizability with customized network architecture.

Among the current optimizers, Adam combines the advantages of momentum on the RMSProp method's base, stabilizing it via additional hyperparameters, such as β_1, β_2, and ϵ.[40] Parameter updates can be made more stable when initializing or encountering small gradients continuously. Nevertheless, Adam tends to have poor generalization than traditional SGD. Even though a fast convergence is accomplished during training, this causes the fact that errors during testing are almost much worse than those during training.[41] Some studies also mention that Adam has convergence problems in some cases. The adaptive learning rate algorithm in Adam may lead to suboptimal solutions due to exponential moving averages.[42] MLP models are more restricted models with redundant architectures than the NE model, so the neural networks trained in such a space with extremely high dimensions easily get stuck in the saddle and stagnate. In our approach, since the model architectures are constantly optimized in addition to the parameter update during each training, the abovementioned dilemma can be overcome during training. Given a problem, the NE model has more chance to smooth the error surface owing to the calibration through GA, that is, an easier way to approach the minimum in comparison with the MLP model.

■ CONCLUSIONS

This work can robustly and automatically tune the model architecture and find the model capable of extracting domain knowledge in semiconductor manufacturing problems. The autoevolved network architecture is shown to be self-adaptive to the domain knowledge, which can be important in many practical fields where data are expensive, or the problems are highly complex. With the combination of NE and Adam, the optimal model architectures can achieve convergence with the train set loss and validation set loss settled at smaller values in reference to MLP baselines. Compared with MLP, overfitting can be avoided, and less data are required for training. After training, the prediction of the test data and the trend graph drawn reflect that the optimal NE model has a better comprehension of the domain knowledge, denoting that the method can reduce the need for complex formulas and tremendous human effort to build appropriate model architectures.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acsomega.3c04123.

Data distribution plots for three dataset sizes including 1920, 2880, and 3840 and different train-test splits (PDF)

■ AUTHOR INFORMATION

Corresponding Author

Albert Shihchun Lin − *Institute of Electronics Engineering, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan;* orcid.org/0000-0001-6104-3360; Email: hdtd5746@gmail.com

Authors

Yen-Wei Feng − *Institute of Electronics Engineering, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan;* orcid.org/0000-0003-0764-2092

Bing-Ru Jiang − *Institute of Electronics Engineering, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan;* orcid.org/0000-0003-2049-045X

Complete contact information is available at: https://pubs.acs.org/10.1021/acsomega.3c04123

Author Contributions

The manuscript was written mainly by Y.-W.F. and revised by Albert Lin. The TCAD simulation is conducted by Y.-W.F., and Python programming is conducted by B.-R.J. A.L. supervises the research. All authors have given approval to the final version of the manuscript. Y.-W.F. and B.-R.J. contributed equally.

Funding

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 111-2221-E-A49-186.

Notes

The authors declare no competing financial interest.

The code is available at https://github.com/albertlin11/NE_TCADds.

■ ABBREVIATIONS

TCAD, technology computer-aided design; MLP, multilayer perceptron; ML, machine learning; NAS, neural architecture search; NE, neuroevolution; HBT, heterojunction bipolar transistor; Ic, collector currents; Vce, collector voltage; GA, genetic algorithms; Adam, adaptive moment estimation; mape, mean absolute percentage error

■ REFERENCES

(1) (a) Çınar, Z. M.; Abdussalam Nuhu, A.; Zeeshan, Q.; Korhan, O.; Asmael, M.; Safaei, B. Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0. *Sustainability* **2020**, *12*, 8211. (b) Ademujimi, T. T.; Brundage, M. P.; Prabhu, V. V. A review of current machine learning techniques used in manufacturing diagnosis. In *Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing: IFIP WG 5.7 International Conference, APMS 2017, Hamburg, Germany, September 3−7, 2017, Proceedings, Part I*; Springer, 2017; pp 407−415. (c) Weichert, D.; Link, P.; Stoll, A.; Rüping, S.; Ihlenfeldt, S.; Wrobel, S. A review of machine learning for the optimization of production processes. *Int. J. Adv. Manuf. Technol.* **2019**, *104*, 1889−1902. (d) Wuest, T.; Weimer, D.; Irgens, C.; Thoben, K.-D. Machine learning in manufacturing: advantages, challenges, and applications. *Prod. Manuf. Res.* **2016**, *4*, 23−45. (e) Salahshoor, K.; Kordestani, M.; Khoshro, M. S. Fault detection and diagnosis of an industrial steam turbine using fusion of SVM (support vector machine) and ANFIS (adaptive neuro-fuzzy inference system) classifiers. *Energy* **2010**, *35*, 5472−5482. (f) Pham, D. T.; Afify, A. A. Machine-learning techniques and their applications in manufacturing. *Proc. Inst. Mech. Eng., Part B* **2005**, *219*, 395−412.

(2) (a) Maggipinto, M.; Terzi, M.; Masiero, C.; Beghi, A.; Susto, G. A. A computer vision-inspired deep learning architecture for virtual metrology modeling with 2-dimensional data. *IEEE Trans. Semicond. Manuf.* **2018**, *31*, 376−384. (b) Nakazawa, T.; Kulkarni, D. V. Anomaly detection and segmentation for wafer defect patterns using deep convolutional encoder−decoder neural network architectures in semiconductor manufacturing. *IEEE Trans. Semicond. Manuf.* **2019**, *32*, 250−256.

(3) Yang, Y.-F.; Sun, M. A Novel Deep Learning Architecture for Global Defect Classification: Self-Proliferating Neural Network (SPNet). In *2021 32nd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*; IEEE, 2021; pp 1−6.

(4) (a) Hsu, C.-Y.; Liu, W.-C. Multiple time-series convolutional neural network for fault detection and diagnosis and empirical study in semiconductor manufacturing. *J. Intell. Manuf.* **2021**, *32*, 823−836. (b) Ghahramani, M.; Qiao, Y.; Zhou, M. C.; O'Hagan, A.; Sweeney, J. AI-based modeling and data-driven evaluation for smart manufacturing processes. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 1026−1037.

(5) (a) He, C.; Hu, H.; Li, P. Applications for Machine Learning in Semiconductor Manufacturing and Test. In *2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*; IEEE, 2021; pp 1−3. (b) Li, T.-S.; Huang, C.-L. Defect spatial pattern recognition using a hybrid SOM−SVM approach in semiconductor manufacturing. *Expert Syst. Appl.* **2009**, *36*, 374−385. (c) Shin, C. K.; Park, S. C. A machine learning approach to yield management in semiconductor manufacturing. *Int. J. Prod. Res.* **2000**, *38*, 4261−4271. (d) Susto; Schirru, A.; Pampuri, S.; McLoone, S.; Beghi, A. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Trans. Ind. Inf.* **2015**, *11*, 812−820.

(6) Elsken, T.; Metzen, J. H.; Hutter, F. Neural architecture search: A survey. *J. Mach. Learn. Res.* **2019**, *20*, 1−21.

(7) Stanley, K. O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99−127.

(8) (a) Galván, E.; Mooney, P. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Trans. Artif. Intell.* **2021**, *2*, 476−493. (b) Ryerkerk, M. L.; Averill, R. C.; Deb, K.; Goodman, E. D. Solving metameric variable-length optimization problems using genetic algorithms. *Genet. Program. Evol. Mach.* **2017**, *18*, 247−277. (c) Liu, Y.; Sun, Y.; Xue, B.; Zhang, M.; Yen, G. G.; Tan, K. C. A survey on evolutionary neural architecture search. *IEEE Trans. Neural Networks Learn. Syst.* **2021**, DOI: 10.1109/TNNLS.2021.3134673.

(9) (a) Zoph, B.; Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. (b) Baker, B.; Gupta, O.; Naik, N.; Raskar, R. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. (c) Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018; pp 8697−8710.

(10) (c) Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; Le, Q. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*; PMLR, 2018; pp 550−559. (b) Liu, H.; Simonyan, K.; Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

(11) (a) Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; Xing, E. P. Neural architecture search with bayesian optimisation and optimal transport. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 7191. (b) Wistuba, M. Bayesian optimization combined with incremental evaluation for neural network architecture optimization. In *Proceedings*

of the International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms, 2017.

(12) (a) Mendoza, H.; Klein, A.; Feurer, M.; Springenberg, J. T.; Hutter, F. Towards automatically-tuned neural networks. In *Workshop on automatic machine learning*; PMLR, 2016; pp 58−65. (b) White, C.; Neiswanger, W.; Savani, Y. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021; Vol. 35, pp 10293−10301. (c) Zela, A.; Klein, A.; Falkner, S.; Hutter, F. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

(13) Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; Murphy, K. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, 2018; pp 19−34.

(14) (a) Camero, A.; Wang, H.; Alba, E.; Bäck, T. Bayesian neural architecture search using a training-free performance metric. *Appl. Soft Comput.* **2021**, *106*, No. 107356. (b) He, X.; Zhao, K.; Chu, X. AutoML: A survey of the state-of-the-art. *Knowl.-Based Syst.* **2021**, *212*, No. 106622. (c) Thornton, C.; Hutter, F.; Hoos, H. H.; Leyton-Brown, K. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013; pp 847−855.

(15) Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281.

(16) Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

(17) Mo, S.; Xia, J.; Ren, P. Simulated Annealing for Neural Architecture Search. In *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*, New Orleans, USA; 2021.

(18) (a) Lang, S.; Reggelin, T.; Schmidt, J.; Müller, M.; Nahhas, A. NeuroEvolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies. *Expert Syst. Appl.* **2021**, *172*, No. 114666. (b) Li, X.; Zheng, J.; Li, M.; Ma, W.; Hu, Y. One-shot neural architecture search for fault diagnosis using vibration signals. *Expert Syst. Appl.* **2022**, *190*, No. 116027. (c) Zhou, J.; Zheng, L.; Wang, Y.; Wang, C.; Gao, R. X. Automated model generation for machinery fault diagnosis based on reinforcement learning and neural architecture search. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 1−12.

(19) Liu, J.; Qiao, F.; Zou, M.; Zinn, J.; Ma, Y.; Vogel-Heuser, B. Dynamic scheduling for semiconductor manufacturing systems with uncertainties using convolutional neural networks and reinforcement learning. *Complex Intell. Syst.* **2022**, *8*, 4641−4662.

(20) Deng, C.; Ji, X.; Rainey, C.; Zhang, J.; Lu, W. Integrating machine learning with human knowledge. *iScience* **2020**, *23*, No. 101656.

(21) Hao, Z.; Liu, S.; Zhang, Y.; Ying, C.; Feng, Y.; Su, H.; Zhu, J. Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications. *arXiv preprint arXiv:2211.08064*, 2022.

(22) Karniadakis, G. E.; Kevrekidis, I. G.; Lu, L.; Perdikaris, P.; Wang, S.; Yang, L. Physics-informed machine learning. *Nat. Rev. Phys.* **2021**, *3*, 422−440.

(23) Xu, Y.; Kohtz, S.; Boakye, J.; Gardoni, P.; Wang, P. Physics-informed machine learning for reliability and systems safety applications: State of the art and challenges. *Reliab. Eng. Syst. Saf.* **2023**, *230*, No. 108900.

(24) (a) Bankapalli, Y.; Wong, H. TCAD augmented machine learning for semiconductor device failure troubleshooting and reverse engineering. In *2019 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*; IEEE, 2019; pp 1−4. (b) Teo, C.-W.; Low, K. L.; Narang, V.; Thean, A. V.-Y. TCAD-enabled machine learning defect prediction to accelerate advanced semiconductor device failure analysis. In *2019 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*; IEEE, 2019; pp 1−4. (c) Wong, H. Y.; Xiao, M.; Wang, B.; Chiu, Y. K.; Yan, X.; Ma, J.; Sasaki, K.; Wang, H.; Zhang, Y. TCAD-machine learning framework for device variation and operating temperature analysis with experimental demonstration. *IEEE J. Electron Devices Soc.* **2020**, *8*, 992−1000.

(25) Ho, Y.-W.; Rawat, T. S.; Yang, Z.-K.; Pratik, S.; Lai, G.-W.; Tu, Y.-L.; Lin, A. Neuroevolution-based efficient field effect transistor compact device models. *IEEE Access* **2021**, *9*, 159048−159058.

(26) (a) Yang, Y.; Sun, M. Semiconductor Defect Pattern Classification by Self-Proliferation-and-Attention Neural Network. *IEEE Trans. Semicond. Manuf.* **2022**, *35*, 16−23. (b) Howard, G. D.; Bull, L.; De Lacy Costello, B.; Gale, E.; Adamatzky, A. Evolving memristive neural networks. *Handbook Memristor Networks* **2019**, 661−690. (c) Howard, G.; Gale, E.; Bull, L.; de Lacy Costello, B.; Adamatzky, A. Evolution of plastic learning in spiking networks via memristive connections. *IEEE Trans. Evol. Comput.* **2012**, *16*, 711−729.

(27) (a) Jiang, S.; Balaprakash, P. Graph neural network architecture search for molecular property prediction. In *2020 IEEE International conference on big data (big data)*; IEEE, 2020; pp 1346−1353. (b) Cerecedo-Cordoba, J. A.; Barbosa, J. J. G.; Terán-Villanueva, J. D.; Frausto-Solís, J.; Flores, J. A. M. Use of neuroevolution to estimate the melting point of ionic liquids. *Int. J. Comb. Optim. Probl. Inf.* **2017**, *8*, 2−9.

(28) *Sentaurus Process User Guide*; Synopsys, Inc., 2016.

(29) *Sentaurus Device User Guide*; Synopsys, Inc., 2016.

(30) Hobler, G.; Selberherr, S. Two-dimensional modeling of ion implantation induced point defects. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* **1988**, *7*, 174−180.

(31) Lever, R.; Bonar, J.; Willoughby, A. Boron diffusion across silicon−silicon germanium boundaries. *J. Appl. Phys.* **1998**, *83*, 1988−1994.

(32) Van Rossum, G.; Drake, F. L., Jr. *The python language reference*; Python Software Foundation: Wilmington, DE, USA, 2014.

(33) Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M. Tensorflow: a system for large-scale machine learning. In *Osdi*; Savannah, GA, USA, 2016; Vol. *16*, pp 265−283.

(34) Harris, C. R.; Millman, K. J.; Van Der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J. Array programming with NumPy. *Nature* **2020**, *585*, 357−362.

(35) McKinney, W. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*; Austin, TX, 2010; Vol. *445*, pp 51−56.

(36) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825−2830.

(37) Higham, D. J.; Higham, N. J. *MATLAB guide*; SIAM, 2016.

(38) Li, M.; İrsoy, O.; Cardie, C.; Xing, H. G. Physics-inspired neural networks for efficient device compact modeling. *IEEE J. Explor. Solid-State Comput. Devices Circ.* **2016**, *2*, 44−49.

(39) Kao, M.-Y.; Kam, H.; Hu, C. Deep-learning-assisted physics-driven MOSFET current-voltage modeling. *IEEE Electron Device Lett.* **2022**, *43*, 974−977.

(40) Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

(41) (a) Keskar, N. S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. (b) Luo, L.; Xiong, Y.; Liu, Y.; Sun, X. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

(42) Reddi, S. J.; Kale, S.; Kumar, S. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.