electronics

Brief Report

A Machine Learning Approach for the Forecasting of Computing Resource Requirements in Integrated Circuit Simulation

Yue Wu, Hua Chen *, Min Zhou and Faxin Yu

School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310000, China
* Correspondence: chenhua@zju.edu.cn

Abstract: For the iterative development of the chip, ensuring that the simulation is completed in the shortest time is critical. To meet this demand, the common practice is to reduce simulation time by providing more computing resources. However, this acceleration method has an upper limit. After reaching the upper limit, providing more CPUs can no longer shorten the simulation time, but will instead waste a lot of computing resources. Unfortunately, the recommended values of the existing commercial tools are often higher than this upper limit. To better match this limit, a machine learning optimization algorithm trained with a custom loss function is proposed. Experimental results demonstrate that the proposed algorithm is superior to commercial tools in terms of both accuracy and stability. In addition, the simulations using the resources predicted by the proposed model maintain the same simulation completion time while reducing core hour consumption by approximately 30%.

Keywords: machine learning; computer resource; integrated circuit simulation; prediction

Citation: Wu, Y.; Chen, H.; Zhou, M.; Yu, F. A Machine Learning Approach for the Forecasting of Computing Resource Requirements in Integrated Circuit Simulation. *Electronics* **2023**, *12*, 95. https://doi.org/10.3390/electronics12010095

Academic Editors: Andrea Prati, Luis Javier García Villalba and Vincent A. Cicirello

Received: 22 November 2022
Revised: 20 December 2022
Accepted: 23 December 2022
Published: 26 December 2022

1. Introduction

Faster time-to-market is critical for integrated circuit products and can seriously affect market returns. Therefore, how to minimize simulation iteration time has always been a top priority. A common practice is to provide more computing resources to boost simulation. However, following Amdahl's law [1], the maximum number of cores used to speed up is limited by the parallelizable proportion of the program. After the provided resources reach the optimal resources, adding more cores can no longer accelerate the simulation. A speedup example for simulation that is 50% partially parallelizable is shown in Figure 1. Therefore, how to accurately predict the optimal resource has become a key issue. In the case of underestimation, the simulation time can be unnecessarily lengthened, whereas overestimation not only leads to a waste of resources but also decreases the number of concurrent simulation tasks in a computing cluster [2], which renders it impossible to simultaneously process massive simulation tasks.

Traditional computing resource prediction relies heavily on the experience of engineers. However, as the structure of integrated circuits becomes more complex and larger, it is difficult for engineers to predict the optimal cores with intuition alone. To address this issue, commercial tools from Cadence, including Spectre X [3], Accelerated Parallel Simulator (APS) [4], and UltraSim [5], have proposed a forecasting method. Nevertheless, the problem of overestimation still exists, as seen in Figure 1.

Fortunately, machine learning has shown great power in predicting the consumption of resources by applications [6]. However, current research (listed in Table 1) focuses primarily on the prediction of resource demands from the perspective of resource providers, particularly the optimization of allocating resources to users to increase resource utilization and provider revenues [7].

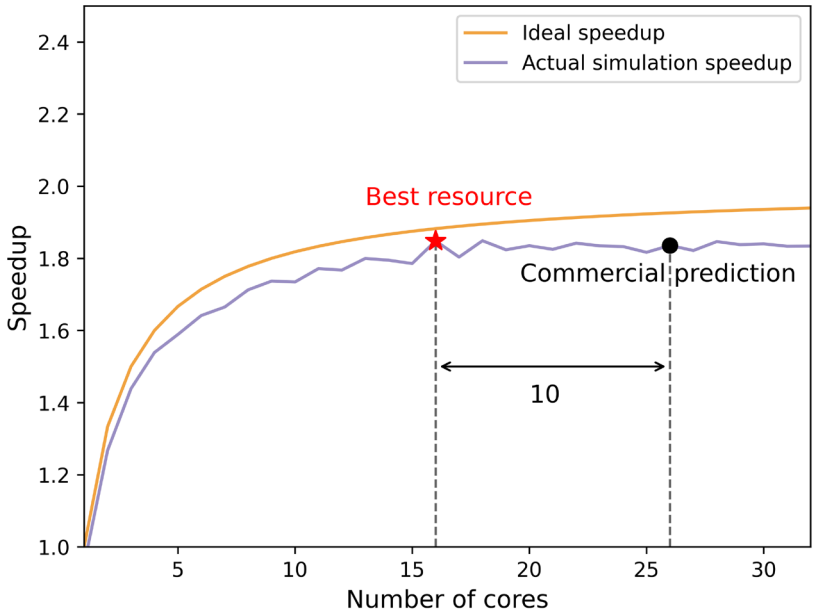


Figure 1. Speedup for simulation that is 50% partially parallelizable.

Table 1. Literature review.

Author	Proposed Methodology	Main Finding
Mahdi Rezaei and Alexey Salnikov (2018) [8]	A machine learning (ML) system based on the collection of statistical data from the reference queue systems.	The accuracy of prediction is highly associated with prior information before submitting jobs.
Mahesh Balaji Philips and Aswani Kumar Cherukuri (2016) [9]	An innovative Predictive Resource Management Framework (PRMF) to overcome the drawbacks of the reactive Cloud Resource Management approach.	Reduced user-request rejections (error count), shorter user-wait time, higher request processing, and efficient utilization of available cloud resources.
Nikravesh et al., (2015) [10]	A predictive auto-scaling system for the Infrastructure as a Service (IaaS) layer of cloud computing.	Achieved dynamic selecting the most appropriate prediction algorithm as well as sliding window size.
Belgacem, Beghdad Bey and Nacer (2020) [11]	A scheduling method based on the Symbiotic Organism Search algorithm (MOSOS) to solve the dynamic resource allocation problem in cloud computing.	Achieved minimized makespan and cost, and adapted to the dynamic change of the cloud.
Vadivel Ramasamy and SudalaiMuthu Thalavai Pillai (2020) [12]	A novel dynamic resource allocation approach based on Hybrid Particle Swarm Optimization and Modified Genetic Algorithm (HPSO-MGA).	Achieved less time for dynamically allocating the resources in terms of average waiting time, response time, load balancing, relative error, and throughput.
Chaitra et.al. (2022) [13]	An independent task scheduling approach using a multi-objective task scheduling optimization based on the Artificial Bee Colony Algorithm with a Q-learning algorithm.	Outperformed in terms of reducing makespan, reducing cost, reducing the degree of imbalance, increasing throughput, and average resource utilization.
Ali Belgacem et al., (2020) [14]	A multi-objective search algorithm called the Spacing Multi-Objective Antlion algorithm (S-MOAL) to minimize both the makespan and the cost of using virtual machines.	Outperformed in terms of makespan, cost, deadline, fault tolerance, and energy consumption.

Although these resource prediction methods improve the overall efficiency of high-performance platforms, they ignore the need to obtain simulation results in the shortest time possible. This paper proposes a machine learning-based model for predicting integrated circuit simulation resource requirements. The contributions of this work include the following:

- Our algorithm outperforms commercial tools in terms of precision and consistency.
- To our knowledge, our method is the first to apply machine learning to predict the optimal resources required for integrated circuit simulations.
- Our model uses the custom Mean Squared Error (MSE) to enhance overall prediction outcomes and reduce simulation time.
- Our method can reduce core hour consumption by approximately 30% compared to commercial tools while maintaining the same completion time.

In this paper, we investigate machine learning models for integrated circuit simulation resource prediction. Implementing machine learning models for resource prediction involves five steps: data collection, data preprocessing, training, prediction, and application. The workflow of developing resource prediction models using machine learning is depicted in Figure 2. Initially, simulation samples comprised of a netlist, simulation options, and optimal computing resources are collected. Then, data preprocessing is performed on the netlist and simulation parameters to obtain model features. After data preprocessing, the k-fold cross-validation method is used to select the optimal hyperparameters for training a neural network. After training, the model estimated the required resources for the simulation in the test set. Finally, the simulation is provided with the predicted resources, and the resource consumption of circuit simulations is analyzed. The article structure as corresponds to the procedure is arranged as follows. In Section 2, the model engine and the data preprocessing for obtaining model features are presented. Section 3 introduces the experiments and discussion, including data collection, training, prediction results, and application effects. Section 4 concludes this paper.

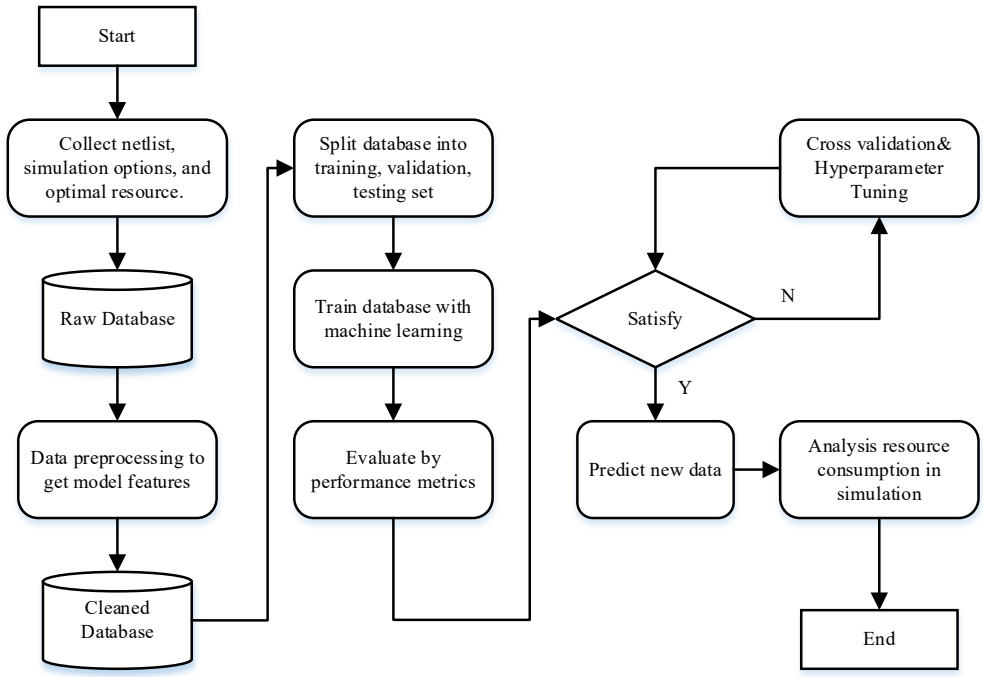


Figure 2. Workflow for implementing machine learning models for resource prediction.

2. Proposed Approach

2.1. Model Feature

According to the circuit simulation mechanism, the simulation-relevant properties can be divided into two parts: the netlist, which describes the connectivity of the circuit, and the simulator options, which influence the simulation process [15]. Consequently, the model features are derived from the netlist and simulator options. In this paper, the distribution of components calculated by a hierarchical tree approach is chosen as a part of the model feature. On the other hand, we analyze the simulator solution process and

select the component parameters and simulation options that influence computing resource requirements as model features.

2.1.1. Component Distribution

In general, circuit simulation consists of pre-layout simulation and post-layout simulation. Unlike the pre-layout simulation netlist, the post-layout simulation netlist contains numerous parasitic elements. Meanwhile, the post-layout simulation requires different computational resources than the pre-layout simulation. Consequently, it is necessary to select the distribution of components as a part of model features.

In large designs, it is common practice to describe circuits using a folded hierarchical netlist. To determine the component distribution, the folded hierarchical netlist is decomposed by the hierarchical tree method. Figure 3 shows a folded hierarchical netlist of a two-stage operational amplifier circuit at the schematic level. The corresponding hierarchical tree is shown in Figure 4.

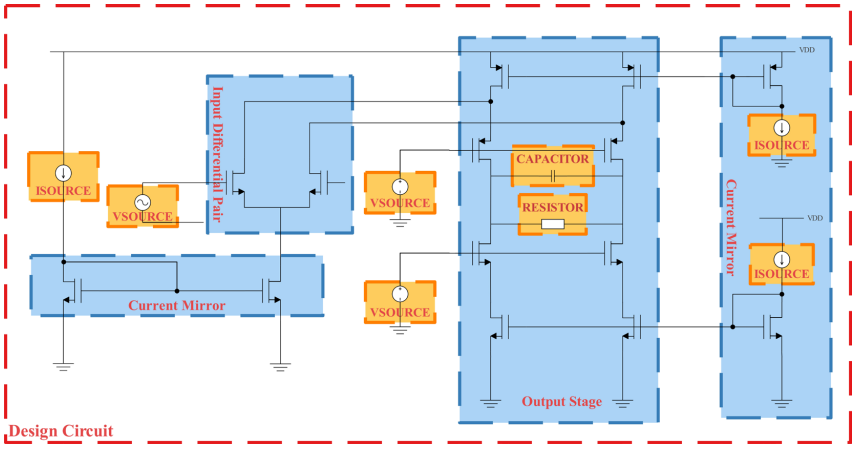


Figure 3. Folded hierarchical netlist.

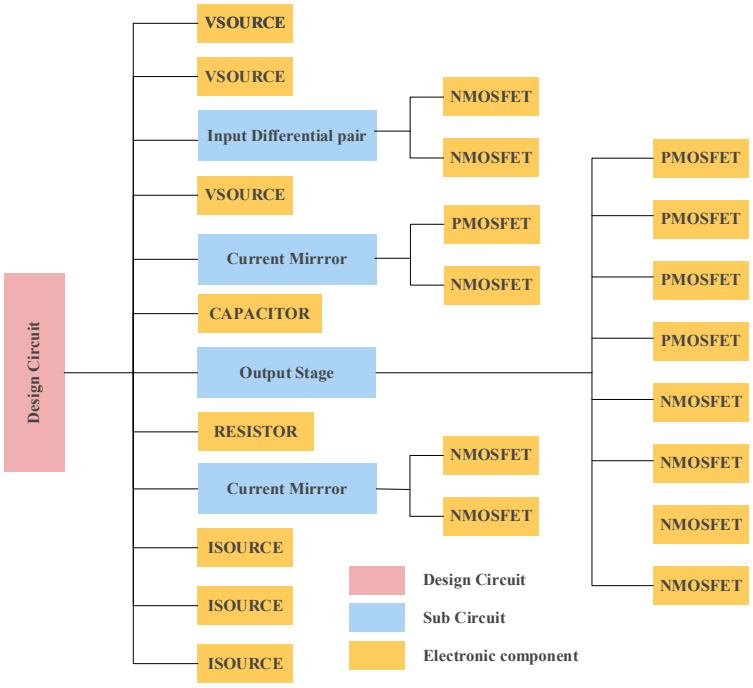


Figure 4. Hierarchical tree.

In the hierarchical tree, the root node represents the entire design circuit, and the leaf nodes represent instances of electronic components. Each leaf node records two characteristics: type and connection. The type indicates the type of component, such as a resistor, capacitor, source, etc. The connection is the internal connection ports of a component, such as a drain and gate of a metal–oxide–semiconductor field-effect transistor are connected together. Considering that the internal connection of components influences the convergence of the circuit, the component with the same type and port connection has become the smallest statistical unit for component distribution. Table 2 displays the component distribution of the hierarchical tree in Figure 4.

Table 2. Component distribution.

Type	Self-Connection	Number
ISOURCE	null	3
VSOURCE	null	3
NMOSFET	null	7
NMOSFET	gate, drain	2
PMOSFET	null	4
PMOSFET	gate, drain	1
RESISTOR	null	1
CAPACITOR	null	1
INDUCTOR	null	0
NDIODE	null	0
PDIODE	null	0
PNP-BJT	null	0
PNP-BJT	emitter, base	0
NPN-BJT	null	0
NPN -BJT	emitter, base	0

2.1.2. Simulation Options

The development of Electronic Design Automation is accompanied by a concomitant proliferation of more emulators, each with a unique naming convention for the parameters. Despite the differing naming regulations, the substantive meaning of the parameters of the same emulator type remains the same. This naturally guarantees that the simulator options selected as model features exist in multiple simulators.

Simulation is the process of solving nonlinear differential-algebraic equations iteratively. The simulator estimates the voltages at each node of a circuit and then calculates the mesh currents using the conductance. The currents are then utilized to recalibrate the voltages at each node, and the iteration is repeated until all of the node voltages have stabilized within predefined tolerance limits. During a transient analysis, the simulator executes this iterative process at each time step until the stop time is reached. If the voltage does not converge at a time step, the simulator will decrease the time step and repeat the iterative process.

Table 3 lists the simulation options that affect the simulation process. The minimum period of the signal source and the start and end time of the simulation determines the change of the signal, which affects the convergence process. The convergence accuracy and error accuracy, as well as the simulator accuracy, directly determine the convergence of the simulation. The time step determines the number of retries during the process. These options contain both categorical and numerical attributes. Categorical labels are converted to numeric values ranging from one to the number of types for the attribute. For example, the simulator accuracy is categorized as liberal, moderate, or conservative. When accuracy is conservative, the feature corresponding to simulator accuracy equals three. For numeric attributes, the attribute value was recorded after being unified into the smallest unit. For example, the time step attribute is unified into femtoseconds, such as one picosecond being converted to one thousand femtoseconds for recording.

Table 3. Simulation options.

Feature	Attributes
Simulator type	categorical
Simulator accuracy	categorical
Error preset accuracy	categorical
Minimum Port source period	numerical
Minimum Voltage source period	numerical
Minimum Current source period	numerical
Relative convergence criterion	numerical
Voltage absolute tolerance convergence criterion	numerical
Current absolute tolerance convergence criterion	numerical
Charge absolute tolerance convergence criterion	numerical
Transient Analysis start time	numerical
Transient Analysis stop time	numerical
Transient Analysis Minimum time step	numerical
Transient Analysis Maximum time step	numerical

2.2. Model Engine

2.2.1. Support Vector Regression

The application of a Support Vector Machine (SVM) to traditional regression analysis is known as Support Vector Regression (SVR). The cost function of SVM is the l2-normed coefficient vector instead of the MSE. In SVM, the error term is processed in the constraints by setting the absolute error to define the specified boundaries. The most important part of SVM lies in the kernel function. The common no-linear kernel function in SVM is the Gaussian Radius Basis function, defined as

$$\mathrm{K}(x, z) = \mathrm{e}^{(-\frac{||x-z||^2}{2\sigma^2})} \tag{1}$$

The σ parameter can be seen as the radius of influence of samples selected, and $||x - z||^2$ is the squared Euclidean distance between two feature vectors x and z.

2.2.2. Neural Network

A neural network consists of an input layer, an output layer, and multiple hidden layers consisting of a set of neurons. The neurons arranged on the hidden layer transform the information from the neurons in the previous layer by performing linear combinations and non-linear activation functions. After the transformation, the network quantifies the loss calculated by the loss function based on the deviation between the output value and the expected value. The backpropagation algorithm is then used to update the weights and biases within the neural network. Following a predetermined number of iterations, model training is complete. A key strength of a neural network is its ability to capture non-linear attributes in data [6]. An additional benefit is that a neural network's loss function can be tailored to meet diverse requirements. In this work, two loss functions are investigated. One is the commonly used standard MSE, defined as

$$\mathrm{J}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

where y_i refers to the best resource and $\hat{y}_i$ is the estimation result calculated by the neural network. A second index for quantifying error is a customized loss function for the prediction of computing resources, which is defined as

$$\mathrm{J} = \mathrm{J}_{MSE} + \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - \hat{y}_i| + y_i - \hat{y}_i}{y_i}\right)^2 \tag{3}$$

Compared to standard MSE, this customized loss function imposes additional penalties in the case of underestimation ($\hat{y}_i < y_i$). The principle is that the increase in simulation time caused by underestimation is unacceptable in comparison to the waste of resources resulting from overestimation. According to Amdahl's law [1], simulations with different parallelizable ratios require different optimal computing resources. The same prediction bias has inconsistent effects on speedup for simulations requiring different computational resources. For example, when the target value is large and the model prediction is slightly smaller than the target, the deviation may have little effect on the acceleration of the simulation. However, when the target value is small and the prediction has the same deviation, this deviation may have a significant effect on the acceleration of the simulation. Therefore, the penalty term is constructed based on the error rate.

2.2.3. Random Forest

A random forest (RF) model is comprised of multiple decision trees, where each tree constructed from independent samples generates predictions based on a multi-level if-then-else rule. To mitigate the instability of each tree, the overall forecast results are produced by averaging the results from each tree in the forest. Compared to the neural network model, the random forest model is considerably more interpretable. However, off-the-shelf random forest engines rarely support custom loss functions.

3. Experiments and Discussion

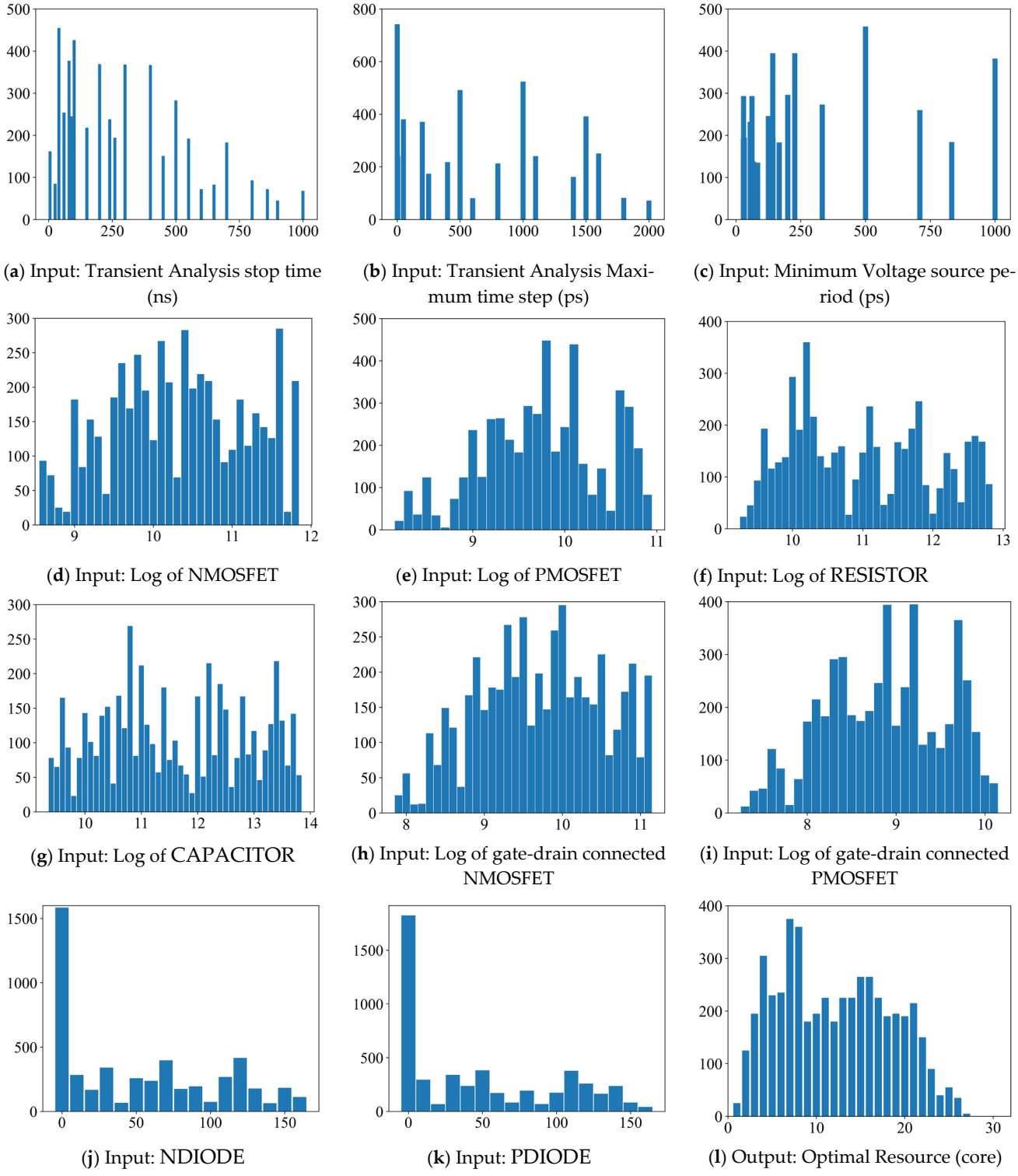
In this research, the SVR and RF models were implemented using the Scikit-Learn library [16], while the neural network was constructed using Keras. Moreover, these machine learning models, the commercial tools for estimating the computational resources required for circuit simulation are compared in the experiment.

3.1. Establishing Dataset

A circuit simulation task consists of the circuit, simulation options, and simulator. In this experiment, the collected circuits were taken from several sources, including standard textbooks [17,18] and papers in the literature [19–24], which are designed using standard foundry analog libraries and commercial CMOS technology. The simulation options are set by circuit design experts; Spectre X, APS, and UltraSim are selected as simulators. A total of five thousand post-layout simulation tasks were gathered. Then, to obtain the model target, varied computing resources are assigned to each simulation task, and the resource with the shortest elapsed time is collected as the optimal computing resource. Finally, we employ the method proposed in Section 2.1 to extract component distribution and resource-related simulation options from the simulation task as model features. To obtain a more comfortable distribution, the natural logarithm is applied to the skewed data. The collected dataset is visualized in Figure 5.

3.2. Model Training

The expressive capacity of a machine learning model is heavily influenced by parameter selection. During the five-fold cross-validation process, the grid search method was used to determine the optimal hyperparameters of the targeted method. For the SVR model, the Gaussian kernel function was used as the kernel function, and the kernel coefficient and regularization parameters were set to 0.0625 and 26, respectively. For the RF model in this research, the number of decision trees and the maximum depth of the tree were selected as 146 and 12, respectively. In the neural network model, the Adam [25] method was employed for optimization, while the Rectified Linear Unit (ReLU) served as the activation function. Using MSE as the loss function, a neural network with 10 hidden layers and eight units per layer achieves the highest level of prediction accuracy. In contrast, when a custom MSE was used as the loss function, the best predictions came from a model where the hidden layer had 8, 8, 6, 6, 4, 4, 4, and 4 units in order.



(**a**) Input: Transient Analysis stop time (ns)

(**b**) Input: Transient Analysis Maximum time step (ps)

(**c**) Input: Minimum Voltage source period (ps)

(**d**) Input: Log of NMOSFET

(**e**) Input: Log of PMOSFET

(**f**) Input: Log of RESISTOR

(**g**) Input: Log of CAPACITOR

(**h**) Input: Log of gate-drain connected NMOSFET

(**i**) Input: Log of gate-drain connected PMOSFET

(**j**) Input: NDIODE

(**k**) Input: PDIODE

(**l**) Output: Optimal Resource (core)

Figure 5. Distribution of collected dataset.

3.3. Prediction Results

Since the number of CPU cores is an integer and the predictions are decimals, the predictions were rounded up to ensure that the simulation could be done as quickly as possible. In accordance with generally accepted standards for measuring model performance, the accuracy of a model is usually determined by two factors: the mean square error (MSE), which indicates how far the predicted values deviate from the target values;

and the correlation coefficient, which indicates how closely the predicted values match the model's input features.

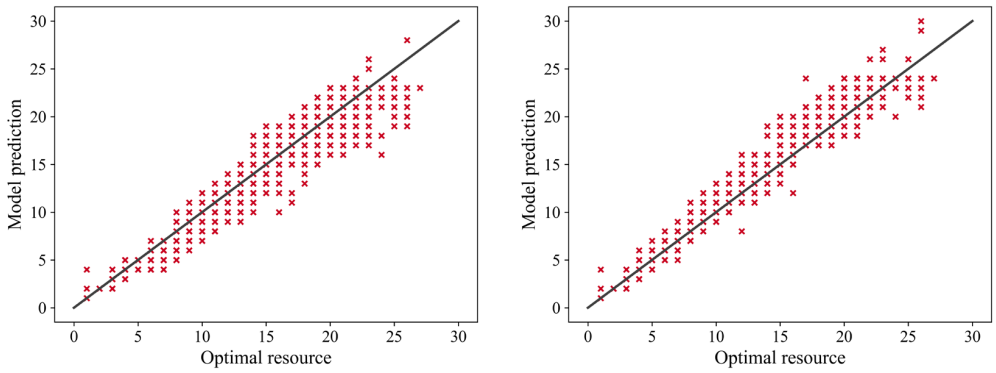
Table 4 displays the performance of the model on the valid dataset during five-fold cross-validation. Among these models, the neural network model employing the standard MSE as a loss function achieved the highest correlation and smallest MSE, whereas the commercial tools achieved the lowest correlation and largest MSE. In addition, the neural network model with custom MSE achieves nearly the same correlation and MSE as the model with standard MSE. The experiment demonstrates that neural network models outperform commercial tools in terms of MSE and correlation coefficients

Table 4. Five-fold cross-validation results.

Model	MSE	Correlation
Commercial tools	29.561	0.521
Random Forest	9.887	0.827
Support Vector Regression	13.891	0.804
Neural network (Standard MSE)	2.163	0.942
Neural network (Custom MSE)	0.942	0.939

3.4. Neural Network with Custom Loss Function

The model accuracy of neural network models with different loss functions is comparable, which is insufficient for selecting the optimal model. In order to further evaluate neural network models with different loss functions, the prediction results of neural network models are examined. The prediction results are shown in Figure 6, where the black line represents the optimal fit between the predictions and target results. It can be seen that the model with custom MSE shifts the forecasts upward, reducing the quantity of underestimated computing resources. Consequently, the model using the custom MSE as the loss function only has 16.7% underestimated predictions, whereas the neural network using the MSE has a 53.4% rate of underestimated predictions.



(**a**) Standard neural network model (**b**) Custom neural network model

Figure 6. Optimal resources versus prediction on the test dataset.

In conclusion, the neural network utilizing the custom MSE yielded nearly identical MSE and correlation, as well as fewer underestimated samples, compared to the model with standard MSE. Considering the need to complete the simulation as quickly as possible, the neural network with the customized MSE is the optimal model.

3.5. Error Distribution

In addition to accuracy, another essential metric for evaluating models is stability, which is typically defined by the distribution of prediction errors. The distribution of prediction errors corresponds to the difference between the predicted and the optimal value. The prediction error is negative when it is underestimated and positive when it is

overestimated. As depicted in Figure 7, the error variances of commercial forecasts are substantial and distributed across positive intervals. In contrast, the error distribution of our method is close to zero and has a smaller variance. This demonstrates that our model is more stable than commercial tools.

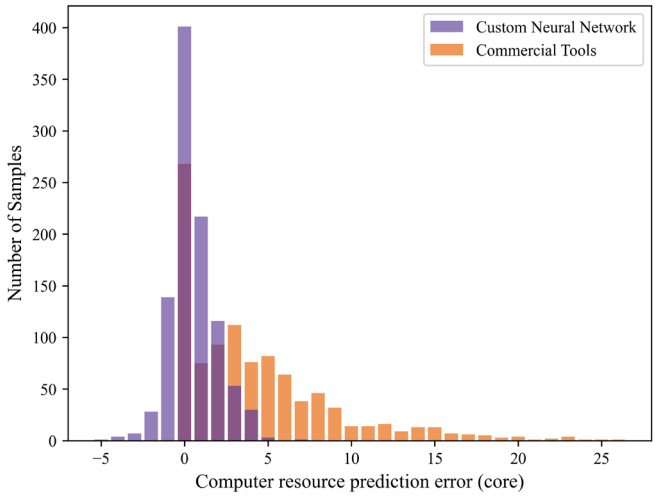
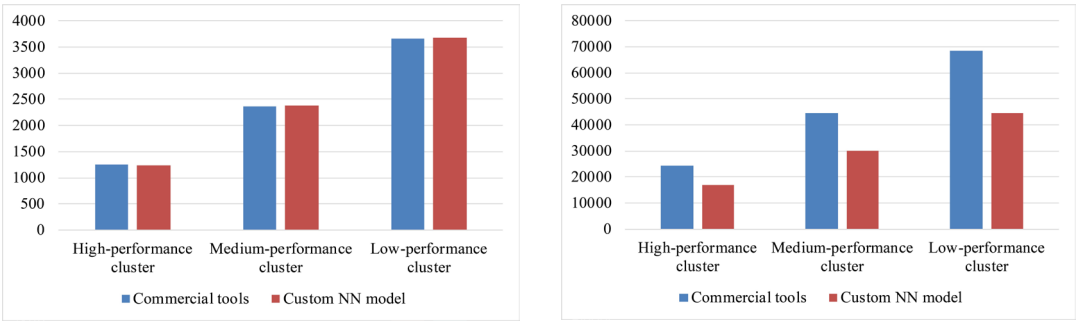


Figure 7. Error distribution in the test dataset.

3.6. Application Effect

The goal of prediction is to reduce redundant computing resources without compromising simulation speed. To verify the ability of the proposed model to accomplish this purpose, three computing environments with varying performance levels were established. Intel Xeon® Gold 6250 processors are utilized in the high-performance computing cluster; Intel Xeon® Gold 6154 processors are utilized in the medium-performance computing cluster; Intel Xeon® Gold 5660 processors are utilized in the low-performance computing cluster. Moreover, the core, each cluster contains 128 GB of memory, a 1 TB SSD, a 10 Gbps NIC, and runs CentOS 7.9.

As shown in Figure 8, the nearly identical simulation elapsed time demonstrates that the proposed model has almost the same effect on the simulation speed as the commercial tools. Meanwhile, compared to the commercial tools, the simulation using the resources predicted by our model consumes no more than 70% of core hours. In summary, compared to the commercial tools, our method can reduce core hours by at least 30% without affecting simulation time.



(**a**) Elapsed time (h) (**b**) Core-hour

Figure 8. Resource consumption of the test dataset in various environments.

4. Conclusions and Future Works

To the best of our knowledge, this paper is the first systematic study of the performance of machine learning algorithms in predicting the computing resources required for circuit simulation. In the selection of model features, simulator options are selected by analyzing the simulation process, and the distribution of components is determined using a hierarchical tree-based method. In addition, a custom loss function is used to improve the performance of the model. Experiments demonstrate that our model outperforms commercial tools in terms of stability and precision. Meanwhile, the application experiments conducted in various computing environments demonstrate that our model can reduce core hours by at least 30% compared to the commercial tools without affecting simulation time.

Due to the variety and complexity of circuits, there is no universal approach for acquiring circuit structure metadata. In future work, we intend to extract the metadata of the circuit structure and utilize it as a model feature to further enhance the accuracy of the model prediction.

Author Contributions: Y.W.: Formal analysis, Investigation. H.C.: Model building, Coding, Writing—original draft. M.Z.: Writing—original draft, Formal analysis. F.Y.: Data collection. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Amdahl, G.M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In Proceedings of the Spring Joint Computer Conference on—AFIPS '67 (Spring), Atlantic City, NJ, USA, 18–20 April 1967; ACM Press: New York, NY, USA, 1967; p. 483.
2. Albers, R.; Suijs, E.; de With, P.H.N. Triple-C: Resource-Usage Prediction for Semi-Automatic Parallelization of Groups of Dynamic Image-Processing Tasks. In Proceedings of the 2009 IEEE International Symposium on Parallel Distributed Processing, Rome, Italy, 23–29 May 2009; pp. 1–8.
3. Spectre X Simulator. Available online: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-x-simulator.html (accessed on 9 December 2022).
4. Spectre Accelerated Parallel Simulator. Available online: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/library-characterization/spectre-accelerated-parallel-simulator.html (accessed on 9 December 2022).
5. Cadence Virtuoso Platform Provides 10x Improvement in Verification Time for VIS. Available online: https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2006/cadencevirtuosoplatformprovides10ximprovementinverificationtimeforvis.html (accessed on 9 December 2022).
6. Matsunaga, A.; Fortes, J.A.B. On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 17–20 May 2010; pp. 495–504.
7. Kholidy, H.A. An Intelligent Swarm Based Prediction Approach For Predicting Cloud Computing User Resource Needs. *Comput. Commun.* **2020**, *151*, 133–144. [CrossRef]
8. Rezaei, M.; Salnikov, A. Machine Learning Techniques to Perform Predictive Analytics of Task Queues Guided by Slurm. In Proceedings of the 2018 Global Smart Industry Conference (GloSIC), Chelyabinsk, Russia, 13–15 November 2018; pp. 1–6.
9. Balaji, M.; Aswani Kumar, C.; Rao, G.S.V.R.K. Predictive Cloud Resource Management Framework for Enterprise Workloads. *J. King Saud Univ.-Comput. Inf. Sci.* **2018**, *30*, 404–415. [CrossRef]
10. Nikravesh, A.Y.; Ajila, S.A.; Lung, C.-H. Towards an Autonomic Auto-Scaling Prediction System for Cloud Resource Provisioning. In Proceedings of the 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Florence, Italy, 18–19 May 2015; pp. 35–45.
11. Belgacem, A.; Beghdad Bey, K.; Nacer, H. Dynamic Resource Allocation Method Based on Symbiotic Organism Search Algorithm in Cloud Computing. *IEEE Trans. Cloud Comput.* **2020**, *10*, 1714–1725. [CrossRef]
12. Ramasamy, V.; Pillai, S. An Effective HPSO-MGA Optimization Algorithm for Dynamic Resource Allocation in Cloud Environment. *Clust. Comput.* **2020**, *23*, 1711–1724. [CrossRef]
13. Chaitra, T.; Agrawal, S.; Jijo, J.; Arya, A. Multi-Objective Optimization for Dynamic Resource Provisioning in a Multi-Cloud Environment Using Lion Optimization Algorithm. In Proceedings of the 2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 5–7 November 2020; pp. 83–90. [CrossRef]

14. Belgacem, A.; Beghdad-Bey, K.; Nacer, H.; Bouznad, S. Efficient Dynamic Resource Allocation Method for Cloud Computing Environment. *Cluster Comput.* **2020**, *23*, 2871–2889. [CrossRef]
15. Barboza, E.C.; Shukla, N.; Chen, Y.; Hu, J. Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism. In Proceedings of the 56th Annual Design Automation Conference 2019 ACM, Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
16. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-Learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
17. Razavi, B.; Behzad, R. *RF Microelectronics*; Prentice Hall: New York, NY, USA, 2012; Volume 2.
18. Razavi, B. *Design of Analog CMOS Integrated Circuits*, 2nd ed.; McGraw-Hill Education: New York, NY, USA, 2017; ISBN 978-0-07-252493-2.
19. Wu, J.; Zhang, K.; Chen, H.; Wang, Z.; Yu, F. A Sigma-Delta Modulator with Residual Offset Suppression. *IEICE Electron. Express* **2021**, *18*, 20210157. [CrossRef]
20. Wang, T.; Zhou, M.; Liu, J.; Wang, Z.; Mo, J.; Chen, H.; Yu, F. A Highly Linear 10 Gb/s MOS Current Mode Logic Driver with Large Output Voltage Swing Based on an Active Inductor. *IEICE Electron. Express* **2020**, *17*, 20200160. [CrossRef]
21. Wang, G.; Liu, J.; Xu, S.; Mo, J.; Wang, Z.; Yu, F. The Design of Broadband LNA with Active Biasing Based on Negative Technique. *Inf. Midem-J. Microelectron. Electron. Compon. Mater.* **2018**, *48*, 115–120.
22. Wang, F.; Wang, Z.; Liu, J.; Yu, F. A 14-Bit 3-GS/s DAC Achieving SFDR >63dB Up to 1.4GHz with Random Differential-Quad Switching Technique. *IEEE Trans. Circuits Syst. II-Express Briefs* **2022**, *69*, 879–883. [CrossRef]
23. Chen, M.; Wu, K.; Shen, Y.; Wang, Z.; Chen, H.; Liu, J.; Yu, F. A 14 bit 500 MS/s 85.62 dBc SFDR 66.29 dB SNDR SHA-Less Pipelined ADC with a Stable and High-Linearity Input Buffer and Aperture-Error Calibration in 40 nm CMOS. *IEICE Electron. Express* **2021**, *18*, 20210171. [CrossRef]
24. Chen, J.; Li, H.; Wang, T.; Wang, Z.; Chen, H.; Liu, J.; Yu, F. A 92 Fs(Rms) Jitter Frequency Synthesizer Based on a Multicore Class-C Voltage-Controlled Oscillator with Digital Automatic Amplitude Control. *IEICE Electron. Express* **2021**, *18*, 20210136. [CrossRef]
25. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization 2017. *arXiv* **2014**, arXiv:1412.6980.