Machine Learning for Electronic Design Automation: A Survey

GUYUE HUANG*, JINGBO HU*, YIFAN HE*, JIALONG LIU*, MINGYUAN MA*, ZHAOYANG SHEN*, JUEJIAN WU*, YUANFAN XU*, HENGRUI ZHANG*, KAI ZHONG*, and XUEFEI NING, Tsinghua University, China
YUZHE MA, HAOYU YANG, and BEI YU, Chinese University of Hong Kong, Hong Kong SAR
HUAZHONG YANG and YU WANG, Tsinghua University, China

With the down-scaling of CMOS technology, the design complexity of very large-scale integrated (VLSI) is increasing. Although the application of machine learning (ML) techniques in electronic design automation (EDA) can trace its history back to the 90s, the recent breakthrough of ML and the increasing complexity of EDA tasks have aroused more interests in incorporating ML to solve EDA tasks. In this paper, we present a comprehensive review of existing ML for EDA studies, organized following the EDA hierarchy.

Additional Key Words and Phrases: electronic design automation, machine learning, neural networks

ACM Reference Format:
Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, and Yu Wang. 2021. Machine Learning for Electronic Design Automation: A Survey. 1, 1 (March 2021), 44 pages.

1 INTRODUCTION

As one of the most important fields in applied computer/electronic engineering, Electronic Design Automation (EDA) has a long history and is still under heavy development incorporating cutting-edge algorithms and technologies. In recent years, with the development of semiconductor technology, the scale of integrated circuit (IC) has grown exponentially, challenging the scalability and reliability of the circuit design flow. Therefore, EDA algorithms and software are required to be more effective and efficient to deal with extremely large search space with low latency.

Machine learning (ML) is taking an important role in our lives these days, which has been widely used in many scenarios. ML methods, including traditional and deep learning algorithms, achieve amazing performance in solving classification, detection, and design space exploration problems. Additionally, ML methods show great potential to generate high-quality solutions for many NP-complete (NPC) problems, which are common in the EDA field, while traditional methods lead to huge time and resource consumption to solve these problems. Traditional methods usually solve every problem from the beginning, with a lack of knowledge accumulation. Instead, ML algorithms focus on extracting high-level features or patterns that can be reused in other related or similar situations, avoiding repeated complicated analysis. Therefore, applying machine learning methods is a promising direction to accelerate the solving of EDA problems.

*These authors are ordered alphabetically.

Authors' addresses: Guyue Huang; Jingbo Hu; Yifan He; Jialong Liu; Mingyuan Ma; Zhaoyang Shen; Juejian Wu; Yuanfan Xu; Hengrui Zhang; Kai Zhong; Xuefei Ning, nxf16@mails.tsinghua.edu.cn, Tsinghua University, China; Yuzhe Ma; Haoyu Yang; Bei Yu, byu@cse.cuhk.edu.hk, Chinese University of Hong Kong, Hong Kong SAR; Huazhong Yang; Yu Wang, yu-wang@tsinghua.edu.cn, Tsinghua University, China.

In recent years, ML for EDA is becoming one of the trending topics, and a lot of studies that use ML to improve EDA methods have been proposed, which cover almost all the stages in the chip design flow, including design space reduction and exploration, logic synthesis, placement, routing, testing, verification, manufacturing, etc. These ML-based methods have demonstrated impressive improvement compared with traditional methods.

We observe that most work collected in this survey can be grouped into four types: *decision making in traditional methods, performance prediction, black-box optimization, and automated design*, ordered by decreasing manual efforts and expert experiences in the design procedure, or an increasing degree of automation. The opportunity of ML in EDA starts from *decision making in traditional methods*, where an ML model is trained to select among available tool chains, algorithms, or hyper-parameters, to replace empirical choice or brute-force search. ML is also used for *performance prediction*, where a model is trained from a database of previously implemented designs to predict the quality of new designs, helping engineers to evaluate new designs without the time-consuming synthesis procedure. Even more automated, EDA tools utilized the workflow of *black-box optimization*, where the entire procedure of design space exploration (DSE) is guided by a predictive ML model and a sampling strategy supported by ML theories. Recent advances in Deep Learning (DL), especially Reinforcement Learning (RL) techniques have stimulated several studies that fully automate some complex design tasks with extremely large design space, where predictors and policies are learned, performed, and adjusted in an online form, showing a promising future of Artificial Intelligence (AI)-assisted *automated design*.

This survey gives a comprehensive review of some recent important studies applying ML to solve some EDA important problems. The review of these studies is organized according to their corresponding stages in the EDA flow. Although the study on ML for EDA can trace back to the last century, most of the works included in this survey are in recent five years. The rest of this survey is organized as follows. In Section 2, we introduce the background of both EDA and ML. From Section 3 to Section 5, we introduce the studies that focus on different stages of the EDA flow, i.e., high-level synthesis, logic synthesis & physical design (placement and routing), and mask synthesis, respectively. In Section 6, analog design methods with ML are reviewed. ML-powered testing and verification methods are discussed in Section 7. Then, in Section 8, other highly-related studies are discussed, including ML for SAT solver and the acceleration of EDA with deep learning engine. The discussion of various studies from the ML perspective is given in Section 9, which is complementary to the main organization of this paper. Finally, Section 10 concludes the existing ML methods for EDA and highlights future trends in this field.

2 BACKGROUND

2.1 Electronic Design Automation

Electronic design automation is one of the most important fields in electronic engineering. In the past few decades, it has been witnessed that the flow of chip design became more and more standardized and complicated. A modern chip design flow is shown in Figure 1.

High-level synthesis (HLS) provides automatic conversion from C/C++/SystemC-based specifications to hardware description languages (HDL). HLS makes hardware design much more convenient by allowing the designer to use high-level descriptions for a hardware system. However, when facing a large-scale system, HLS often takes a long time to finish the synthesis. Consequently, efficient design space exploration (DSE) strategy is crucial in HLS [74, 95, 107, 112, 180].

Logic synthesis converts the behavioral level description to the gate level description, which is one of the most important problems in EDA. Logic synthesis implements the specific logic functions by generating a combination of gates selected in a given cell library, and optimizes the design for

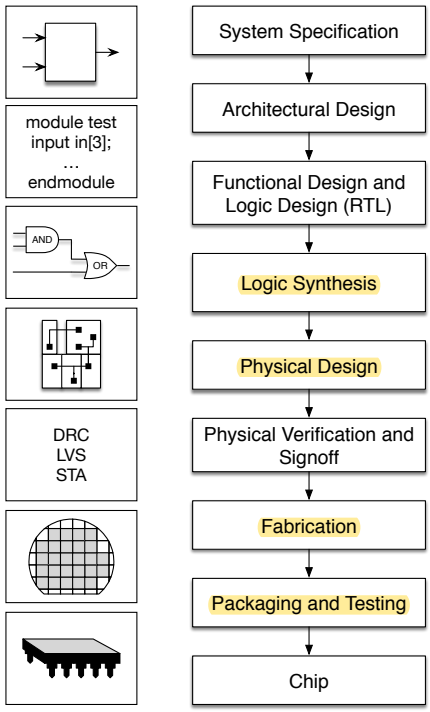


Fig. 1. Modern chip design flow.

different optimization goals. Logic synthesis is a complicated process that usually cannot be solved optimally, and hence the heuristic algorithms are widely used in this stage, which include lots of ML methods [48, 56, 115, 167].

Based on the netlist obtained from synthesis, floorplanning and placement aim to assign the netlist components to specific locations on the chip layout. Better placement assignment implies the potential of better chip area utilization, timing performance, and routability. Routing is one of the essential steps in very large-scale integrated (VLSI) physical design flow based on the placement assignment. Routing assigns the wires to connect the components on the chip. At the same time, routing needs to satisfy the requirements of timing performance and total wirelength without violating the design rules. The placement and routing are strongly coupled. Thus it is crucial to consider the routing performance even in the placement stage, and many ML-based routing-aware methods are proposed to improve the performance of physical design [6, 27, 89, 106, 150, 154].

Fabrication is a complicated process containing multiple steps, which has a high cost in terms of time and resources. Mask synthesis is one of the main steps in the fabrication process, where lithography simulation is leveraged to reduce the probability of fabrication failure. Mask optimization and lithography simulation are still challenging problems. Recently, various ML-based methods are applied in the lithography simulation and mask synthesis [20, 43, 159, 163, 165].

To ensure the correctness of a design, we need to perform design verification before manufacturing. In general, verification is conducted after each stage of the EDA flow, and the test set design is one of the major problems. Traditional random or automated test set generation methods are far away from optimal, therefore, there exist many studies that apply ML methods to optimize test set generation for verification [24, 33, 38, 47, 49, 57, 69, 135, 145, 146].

After the chip design flow is finished, manufacturing testing needs to be carried out. The chips need to go through various tests to verify their functionality and reliability. The coverage and the efficiency are two main optimization goals of the testing stage. Generally speaking, a large test set (i.e., a large number of test points) leads to higher coverage at the cost of high resource consumption. To address the high cost of the testing process, studies have focused on applying ML techniques for test set optimization [100, 120, 140, 141] and test complexity reduction [5, 34, 142].

Thanks to decades of efforts from both academia and industry, the chip design flow is well-developed. However, with the huge increase in the scale of integrated circuits, more efficient and effective methods need to be incorporated to reduce the design cost. Recent advancements in machine learning have provided a far-reaching data-driven perspective for problem-solving. In this survey, we review recent learning-based approaches for each stage in the EDA flow and also discuss the ML for EDA studies from the machine learning perspective.

2.2 Machine Learning

Machine learning is a class of algorithms that automatically extract information from datasets or prior knowledge. Such a data-driven approach is a supplement to analytical models that are widely used in the EDA domain. In general, ML-based solutions can be categorized according to their learning paradigms: **supervised learning**, **unsupervised learning**, **active learning**, and **reinforcement learning**. The difference between supervised and unsupervised learning is whether or not the input data is labeled. With supervised or unsupervised learning, ML models are trained on static data sets offline and then deployed for online inputs without refinement. With active learning, ML models subjectively choose samples from input space to obtain ground truth and refine themselves during the searching process. With reinforcement learning, ML models interact with the environment by taking actions and getting rewards, with the goal of maximizing the total reward. These paradigms all have been shown to be applied to the EDA problems.

As for the model construction, conventional machine learning models have been extensively studied for the EDA problems, especially for physical design [66, 178]. Linear regression, random forest (RF) [91] and artificial neural networks (ANN) [55] are classical regression models. Support vector machine (SVM) [12] is a powerful classification algorithm especially suitable for tasks with a small size of training set. Other common classification models include K-Nearest-Neighbor (KNN) algorithm [39] and RF. These models can be combined with ensemble or boosting techniques to build more expressive models. For example, XGBoost [23] is a gradient boosting framework frequently used in the EDA problems.

Thanks to large public datasets, algorithm breakthrough, and improvements in computation platforms, there have been efforts of applying deep learning (DL) for EDA. In particular, popular models in recent EDA studies include convolutional neural network (CNN) [37, 111], recurrent neural networks (RNN) [83, 148], generative adversarial network (GAN) [165], deep reinforcement learning (DRL) [113, 147] and graph neural networks (GNN) [147, 168]. CNN models are composed of convolutional layers and other basic blocks such as non-linear activation functions and down-sample pooling functions. While CNN is suitable for feature extraction on grid structure data like 2-D image, RNN is good at processing sequential data such as text or audio. GNN is proposed for data organized as graphs. GAN trains jointly a generative network and a discriminative network which compete against each other to eventually generate high quality fake samples. DRL is a class of algorithms that incorporated deep learning into the reinforcement learning paradigm, where an agent learns a strategy from the rewards acquired with previous actions to determine the next action. DRL has achieved great success in complicated tasks with large decision space (e.g., Go game [138]).

3 HIGH LEVEL SYNTHESIS

High-level synthesis (HLS) tools provide automatic conversion from C/C++/SystemC-based specification to hardware description languages like Verilog or VHDL. HLS tools developed in industry and academia [1, 2, 15] have greatly improved productivity in customized hardware design. High-quality HLS designs require appropriate pragmas in the high-level source code related to parallelism, scheduling and resource usage, and careful choices of synthesis configurations in post-Register-Transfer-Level (RTL) stage. Tuning these pragmas and configurations is a non-trivial task, and the long synthesis time for each design (hours from the source code to the final bitstream) prohibits exhaustive DSE.

ML techniques have been applied to improve HLS tools from the following three aspects: fast and accurate result estimation [30, 37, 108, 109, 143, 164, 172], refining conventional DSE algorithms [74, 107, 149], and reforming DSE as an active-learning problem [94, 95, 112, 180]. In addition to achieving good results on individual problems, previous studies have also introduced new generalizable techniques about feature engineering [30, 108, 109, 164, 172], selection and customization of ML models [143], and design space sampling and searching strategies [95, 112, 180].

This section is organized as follows. Section 3.1 introduces recent studies on employing ML for result estimation, often in a static way. Section 3.2 introduces recent studies on adopting ML in DSE workflow, either to improve conventional methods or in the form of active learning.

3.1 Machine Learning for Result Estimation

The reports from HLS tools provide important guidance for tuning the high-level directives. However, acquiring accurate result estimation in an early stage is difficult due to complex optimizations in the physical synthesis, imposing a trade-off between accuracy (waiting for post-synthesis results) and efficiency (evaluating in the HLS stage). ML can be used to improve the accuracy of HLS reports through learning from real design benchmarks. In Section 3.1.1, we introduce previous work on predicting the timing, resource usage, and operation delay of an HLS design. In Section 3.1.2 we describe two types of research about cross-platform performance prediction.

3.1.1 Estimation of Timing, Resource Usage, and Operation Delay. The overall workflow of timing and resource usage prediction is concluded in Figure 2. This workflow is first proposed by Dai et al. [30] and augmented by Makrani et al. [108] and Ferianc et al. [37]. The main methodology is to train an ML model that takes HLS reports as input and outputs a more accurate implementation report without conducting the time-consuming post-implementation. The workflow proposed by Dai et al. [30] can be divided into two steps: *data processing* and *training estimation models*.

Step 1: Data Processing. To enable ML for HLS estimation, we need a dataset for training and testing. The HLS and implementation reports are usually collected across individual designs by running each design through the complete C-to-bitstream flow, for various clock periods and targeting different FPGA devices. After that, one can extract features from the HLS reports as inputs and features from implementation reports as outputs. Besides, to overcome the effect of colinearity and reduce the dimension of the data, previous studies often apply feature selection techniques to systematically remove unimportant features. The most commonly used features are summarized in Table 1.

Step 2: Training Estimation Models. After constructing the dataset, regression models are trained to estimate post-implementation resource usages and clock periods. Frequently used metrics to report the estimation error include relative absolute error (RAE) and relative root mean squared error (RMSE). For both metrics, lower is better. RAE is defined in Equation (1), where $\hat{y}$ is a vector of values predicted by the model, y is a vector of actual ground truth values in the testing set, and

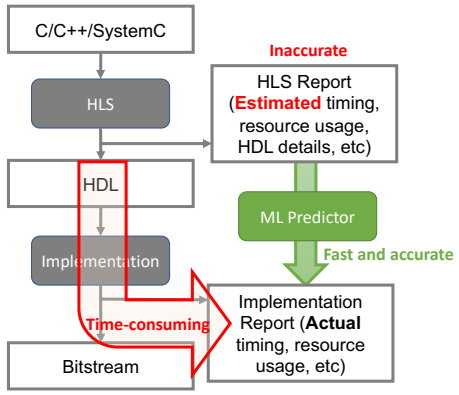


Fig. 2. FPGA tool flow with HLS, highlighting ML-based result predictor (reproduced from [30]).

Table 1. Categories of selected features and descriptions [30, 108]

Category	Brief Description
Clock periods	Target clock period; achieved clock period & its uncertainty.
Resources	Utilization and availability of LUT, FF, DSP, and BRAM.
Logic Ops	Bitwidth/resource statistics of operations.
Arithmetic Ops	Bitwidth/resource statistics of arithmetic operations.
Memory	Number of memory words/banks/bits; resource usage for memory.
Multiplexer	Resource usage for multiplexers; multiplexer input size/bitwidth.

$\bar{y}$ denotes the mean value of y.

$$\text{RAE} = \frac{|\hat{y} - y|}{|y - \bar{y}|}. \tag{1}$$

Relative RMSE is given by Equation (2), where N is the number of samples, and $\hat{y}_i$ and y_i are the predicted and actual values of a sample, respectively.

$$\text{Relative RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\hat{y}_i - y_i}{y_i}\right)^2} \times 100\%. \tag{2}$$

Makrani et al. [108] model timing as a regression problem, and use the Minerva tool [36] to obtain results in terms of maximum clock frequency, throughput, and throughput-to-area ratio for the RTL code generated by the HLS tool. Then an ensemble model combining linear regression, neural network, SVM, and random forest, is proposed to conduct estimation and achieve an accuracy higher than 95%. There are also studies that predict whether a post-implementation is required or not, instead of predicting the implementation results. As a representative study, Liu and Schäfer [94] train a predictive model to avoid re-synthesizing each new configuration.

ML techniques have been applied recently to reduce the HLS tool's prediction error of the operation delay [143]. Existing HLS tools perform delay estimations based on the simple addition of pre-characterized delays of individual operations, and can be inaccurate because of the post-implementation optimizations (e.g., mapping to hardened blocks like DSP adder cluster). A customized Graph Neural Network (GNN) model is built to capture the association between operations from the dataflow graph, and train this model to infer the mapping choices about hardened blocks. Their method can reduce the RMSE of the operation delay prediction of Vivado HLS by 72%.

3.1.2 Cross-Platform Performance Prediction. Hardware/software co-design enables designers to take advantage of new hybrid platforms such as Zynq. However, dividing an application into two parts makes the platform selection difficult for the developers, since there is a huge variation in the application's performance of the same workload across various platforms. To avoid fully implementing the design on each platform, Makrani et al. [109] propose an ML-based cross-platform performance estimator, XPPE, and its overall workflow is described in Figure 3. The key functionality of XPPE is using the resource utilization of an application on one specific FPGA to estimate its performance on other FPGAs.

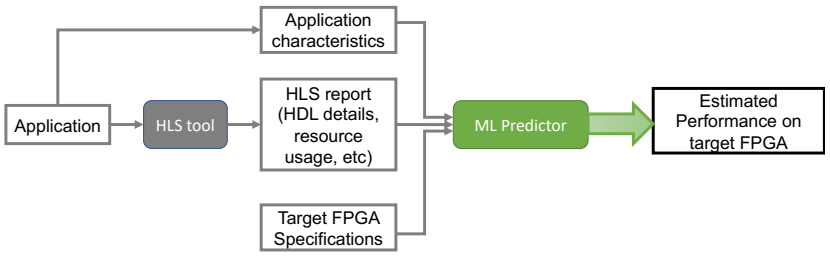


Fig. 3. Overall workflow of XPPE (reproduced from [109]).

XPPE uses a Neural Network (NN) model to estimate the speedup of an application for a target FPGA over the ARM processor. The inputs of XPPE are available resources on target FPGA, resource utilization report from HLS Vivado tool (extracted features, similar to the features in Table 1), and application's characteristics. The output is the speedup estimation on the target FPGA over an ARM A-9 processor. This method is similar to Dai et al. [30] and Makrani et al. [108] in that they all take the features in HLS reports as input and aim to avoid the time-consuming post-implementation. The main difference is that the input and output features in XPPE are from different platforms. The relative RMSE between the predictions and the real measurements is used to evaluate the accuracy of the estimator. The proposed architecture can achieve a relative mean square error of 5.1% and the speedup is more than 0.98×.

Like XPPE, O'Neal et al. [116] also propose an ML-based cross-platform estimator, named HLSPredict. There are two differences. First, HLSPredict only takes workloads (the applications in XPPE) as inputs instead of the combination of HLS reports, application's characteristics and specification of target FPGA devices. Second, the target platform of HLSPredict must be the same as the platform in the training stage. In general, HLSPredict aims to rapidly estimate performance on a specific FPGA by direct execution of a workload on a commercially available off-the-shelf host CPU, but XPPE aims to accurately predict the speedup of different target platforms. For optimized workloads, HLSPredict achieves a relative absolute percentage error ($APE = |\frac{y-\hat{y}}{y}|$) of 9.08% and a 43.78× runtime speedup compared with FPGA synthesis and direct execution.

3.2 Machine Learning for Design Space Exploration in HLS

In the previous subsection, we describe how ML models are used to predict the quality of results. Another application of ML in HLS is to assist DSE. The tunable synthesis options in HLS, provided in the form of pragmas, span a very large design space. Most often, the task of DSE is to find the Pareto Frontier Curve, on which every point is not fully dominated by any other points under all the metrics.

Classical search algorithms have been applied in HLS DSE, such as Simulated Annealing (SA) and Genetic Algorithm (GA). But these algorithms are unable to learn from the database of previously

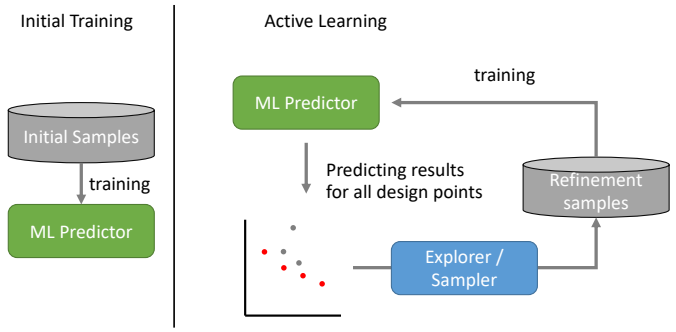


Fig. 4. The iterative-refinement DSE framework (reproduced from [95]).

explored designs. Many previous studies use an ML predictive model to guide the DSE. The models are trained on the synthesis results of explored design points, and used to predict the quality of new designs (See more discussions on this active learning workflow in Section 9.1). Typical studies are elaborated in Section 3.2.1. There is also a thread of work that involves learning-based methods to improve the inefficient or sensitive part of the classical search algorithms, as elaborated in Section 3.2.2. Some work included in this subsection focuses on system-level DSE rather than HLS design [74], or general active learning theories [180].

3.2.1 Active Learning. The four papers visited in this part utilize the active learning approach to perform DSE for HLS, and use predictive ML models to surrogate actual synthesis when evaluating a design. Liu and Schäfer [94] propose a design space explorer that selects new designs to implement through an active learning approach. Transductive experimental design (TED) [95] focuses on seeking the samples that describe the design space accurately. Pareto active learning (PAL) in [180] is proposed to sample designs which the learner cannot clearly classify. Instead of focusing on how accurately the model describes the design space, adaptive threshold non-pareto elimination (ATNE) [112] estimates the inaccuracy of the learner and achieves better performance than TED and PAL.

Liu and Schäfer [94] propose a dedicated explorer to search for Pareto-optimal HLS designs for FPGAs. The explorer iteratively selects potential Pareto-optimal designs to synthesize and verify. The selection is based on a set of important features, which are adjusted during the exploration. The proposed method runs 6.5× faster than an exhaustive search, and runs 3.0× faster than a restricted search method but finds results with higher quality.

The basic idea of TED [95] is to select representative as well as the hard-to-predict samples from the design space, instead of the random sample used in previous work. The target is to maximize the accuracy of the predictive model with the fewest training samples. The authors formulate the problem of finding the best sampling strategy as follows: TED assumes that the overall number of knob settings is $n(|\mathcal{K}| = n)$, from which we want to select a training set $\tilde{\mathcal{K}}$ such that $|\tilde{\mathcal{K}}| = m$. Minimizing the prediction error $H(k) - \tilde{H}(k)$ for all $k \in \mathcal{K}$ is equivalent to the following problem:

$$\max_{\tilde{\mathcal{K}}} T[\mathcal{K}\tilde{\mathcal{K}}^T(\tilde{\mathcal{K}}\tilde{\mathcal{K}}^T + \mu I)^{-1}\tilde{\mathcal{K}}\mathcal{K}^T] \; s.t. \; \tilde{\mathcal{K}} \subset \mathcal{K}, |\tilde{\mathcal{K}}| = m,$$

where $T[\cdot]$ is the matrix trace operator and $\mu > 0$. The authors interpret their solution as sampling from a set $\tilde{\mathcal{K}}$ that *span a linear space, to retain most of the information of* $\mathcal{K}$ [95].

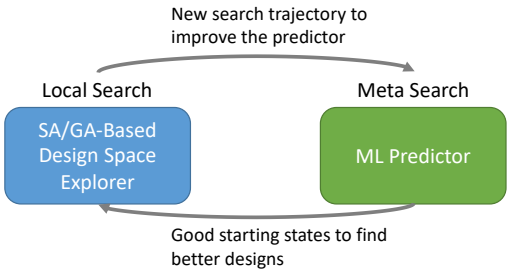


Fig. 5. Overview of the STAGE algorithm (reproduced from [74]).

PAL [180] is proposed for general active learning scenarios and is demonstrated by a sorting network synthesis DSE problem in the paper. It uses Gaussian Process (GP) to predict Pareto-optimal points in design space . The models predict the objective functions to identify points that are Pareto-optimal with high probabilities. A point x that has not been sampled is predicted as $\hat{f}(x) = \mu(x)$ and $\sigma(x)$ is interpreted as the uncertainty of the prediction which can be captured by the hyperrectangle

$$Q_{\mu,\sigma,\beta}(x) = \left\{ y : \mu(x) - \beta^{1/2}\sigma(x) \preceq y \preceq \mu(x) + \beta^{1/2}\sigma(x) \right\},$$

where β is a scaling parameter to be chosen. PAL focuses on accurately predicting points near the Pareto frontier, instead of the whole design space. In every iteration, the algorithm classifies samples into three groups: Pareto-optimal, Non-Pareto-optimal, and uncertain ones. The next design point to evaluate is the one with the largest uncertainty, which intuitively has more information to improve the model. The training process is terminated when there are no uncertain points. The points classified as Pareto-optimal are then returned.

ATNE [112] utilizes Random Forest (RF) to aid the DSE process. This work uses a Pareto identification threshold that adapts to the estimated inaccuracy of the RF regressor and eliminates the non-Pareto-optimal designs incrementally. Instead of focusing on improving the accuracy of the learner, ATNE focuses on estimating and minimizing the risk of losing "good" designs due to learning inaccuracy.

3.2.2 Machine Learning for Improving Other Optimization Algorithms. In this part, we summarize three studies that use ML techniques to improve classical optimization algorithms.

STAGE [74] is proposed for DSE of many-core systems. The motivating observation of STAGE is that the performance of simulated annealing is highly sensitive to the starting point of the search process. The authors build an ML model to learn which parts of the design space should be focused on, eliminating the times of futile exploration [13]. The proposed strategy is divided into two stages. The first stage (local search) performs a normal local search, guided by a cost function based on the designer's goals. The second stage (meta search) tries to use the search trajectories from previous local search runs to learn to predict the outcome of local search given a certain starting point [74].

Fast Simulated Annealing (FSA) [107] utilizes the decision tree to improve the performance of SA. Decision tree learning is a widely used method for inductive inference. The HLS pragmas are taken as input features. FSA first performs standard SA to generate enough training sets to build the decision tree. Then it generates new design configurations with the decision tree and keeps the dominating designs [107].

In a recent study, Wang and Schäfer [149] propose several ML techniques to help decide the hyper-parameter settings of three meta-heuristic algorithms: SA, GA and Ant Colony Optimizations

Table 2. Summary of ML for HLS

Task	Task Details	ML Algorithm	Reference
Result prediction	Timing and resource usage prediction	Lasso, ANN, XGBoost	[30]
	Max frequency, throughput, area	Ridge regression, ANN, SVM, Random Forest	[108]
	Latency	Gaussian Process	[37]
	Operation delay	Graph Neural Network	[143]
Cross-platform	Predict for new FPGA platforms	ANN	[109]
	Predict for new applications through executing on CPUs	Linear models, ANN, Random Forest	[116]
Active learning	Reduce prediction error with fewer samples	Random Forest, Gaussian Process Regression	[95]
	Reduce prediction error for points near the Pareto-frontier	Gaussian Process	[180]
	Reduce the risk of losing Pareto designs	Random Forest	[112]
Improving conventional algorithms	Initial point selection	Quadratic regression	[74]
	Generation of new sample	Decision Tree	[107]
	Hyper-parameter selection	Decision Tree	[149]

(ACO). For each algorithm, the authors build an ML model that predicts the resultant design quality (measured by Average Distance to the Reference Set, ADRS) and runtime from hyper-parameter settings. Compared with the default hyper-parameters, their models can improve the ADRS by more than 1.92× within similar runtime. The authors also combine SA, GA and ACO to build a new design space explorer, which further improves the search efficiency.

3.3 Summary of Machine Learning for HLS

This section reviews recent work on ML techniques in HLS, as listed in Table 2. Using ML-based timing/resource/latency predictors and data-driven searching strategies, the engineering productivity of HLS tools can be further improved and higher-quality designs can be generated by efficiently exploring a large design space.

We believe the following practice can help promote future research of ML in HLS:

- Public benchmark for DSE problems. The researches about result estimation are all evaluated on public benchmarks of HLS applications, such as Rosetta [174], MachSuite [127], etc. However, DSE researches are often evaluated on a few applications because the cost of synthesizing a large design space for each application is heavy. Building a benchmark that collects different implementations of each application can help fairly evaluate DSE algorithms.
- Customized ML models. Most of the previous studies use off-the-shelf ML models. Combining universal ML algorithms with domain knowledge can potentially improve the performance of the model. For example, Ustun et al. [143] customize a standard GNN model to handle the specific delay prediction problem, which brings extra benefit in model accuracy.

4 LOGIC SYNTHESIS AND PHYSICAL DESIGN

In the logic synthesis and physical design stage, there are many key sub-problems that can benefit from the power of ML models, including lithography hotspot detection, path classification, congestion prediction, placement guide, fast timing analysis, logic synthesis scheduling, and so on. In this section, we organize the review of studies by their targeting problems.

4.1 Logic Synthesis

Logic synthesis is an optimization problem with complicated constraints, which requires accurate solutions. Consequently, using ML algorithms to directly generate logic synthesis solutions is difficult. However, there are some studies using ML algorithms to schedule existing traditional optimization strategies. For logic synthesis, LSOracle [115] relies on DNN to dynamically decide which optimizer should be applied to different parts of the circuit. The framework exploits two optimizers, and-inverter graph (AIG) and majority-inverter graph (MIG), and applies k-way partitioning on circuit directed acyclic graph (DAG).

There are many logic transformations in current synthesis tools such as ABC [14]. To select an appropriate synthesis flow, Yu et al. [167] formulate a multi-class classification problem and design a CNN to map a synthesis flow to quality of results (QoR) levels. The prediction on unlabeled flows are then used to select the optimal synthesis flow. The CNN takes the one-hot encoding of synthesis flows as inputs and outputs the possibilities of the input flow belonging to different QoR metric levels.

Reinforcement learning is also employed for logic synthesis in [48, 56]. A transformation between two DAGs with the same I/O behaviors is modeled as an action. In [48], GCN is utilized as a policy function to obtain the probabilities for every action. [56] employs advantage actor critic agent (A2C) to search the optimal solution.

4.2 Placement and Routing Prediction

4.2.1 Traditional Placers Enhancement. While previous fast placers can conduct random logic placement efficiently with good performances, researchers find that their placement of data path logic is suboptimal. PADE [150] proposes a placement process with automatic data path extraction and evaluation, in which the placement of data path logic is conducted separately from random logic. PADE is a force-directed global placer, which applies SVM and NN to extract and evaluate the data path patterns with high dimensional data such as netlist symmetrical structures, initial placement hints, and relative area. The extracted data path is mapped to bit stack structure and uses SAPT [151] (a placer placed on SimPL [73]) to optimize separately from random logic.

4.2.2 Routing Information Prediction. The basic requirements of routing design rules must be considered in the placement stage. However, it is difficult to predict routing information in the placement stage accurately and fast, and researchers recently employ machine learning to solve this. RouteNet [154] is the first work to employ CNN for design rule checking (DRC) hotspot detection. The input features of a customized fully convolutional network (FCN) include the outputs of rectangular uniform wire density (RUDY), a pre-routing congestion estimator. An 18-layer ResNet is also employed to predict design rule violation (DRV) count. A recent work [89] abstracts the pins and macros density in placement results into image data, and utilizes a pixel-wise loss function to optimize an encoder-decoder model (an extension of U-Net architecture). The network output is a heat-map, which represents the location where detailed routing congestion may occur. PROS [21] takes advantages of fully convolution networks to predict routing congestion from global placement results. The framework is demonstrated efficient on industrial netlists. Pui et al. [124] explore the possibilities using ML methods to predict routing congestion in UltraScale FPGAs. Alawieh et al.

[6] transfer the routing congestion problem in large-scale FPGAs to an image-to-image problem, and then uses conditional GAN to solve it. In addition, there are some studies that only predict the number of congestions instead of the location of congestion [27, 106]. Maarouf et al. [106] use models like linear regression, RF and MLP to learn how to use features from earlier stages to produce more accurate congestion prediction, so that the placement strategy can be adjusted. Qi et al. [125] predict the detailed routing congestion using nonparametric regression algorithm, multivariate adaptive regression splines (MARS) with the global information as inputs. Another study [18] takes the netlist, clock period, utilization, aspect ratio and BEOL stack as inputs and utilizes MARS and SVM to predict the routability of a placement. This study also predicts Pareto frontiers of utilization, number of metal layers, and aspect ratio. Study in [99] demonstrates the potential of embedding ML-based routing congestion estimator into global placement stage. Recently, Liang et al. [90] build a routing-free crosstalk prediction model by adopting several ML algorithms such as regression, NN, GraphSAGE and GraphAttention. The proposed framework can identify nets with large crosstalk noise before the routing step, which allows us to modify the placement results to reduce crosstalk in advance.

There is also a need to estimate the final wirelength, timing performance, circuit area, power consumption, clock and other parameters in the early stage. Such prediction task can be modeled as a regression task and commonly-used ML models include SVM, Boosting, RF, MARS, etc. Jeong et al. [63] learn a model with MARS to predict performance from a given set of circuit configurations, with NoC router, a specific functional circuit and a specific business tool. In [60], the researchers introduce linear discriminant analysis (LDA) algorithm to find seven combined features for the best representation, and then a KNN-like approach is adopted to combine the prediction results of ANN, SVM, LASSO, and other machine learning models. In this way, Hyun et al. [60] improve the wirelength prediction given by the virtual placement and routing in the synthesis. Cheng et al. [27] predict the final circuit performance in the macro placement stage, and Li and Franzon [82] predict the circuit performance in the global routing stage, including congestion number, hold slack, area and power.

For sign-off timing analysis, Barboza et al. [8] use random forest to give the sign-off timing slack from hand-crafted features. Another research [67] works on sign-off timing analysis and use linear regression to fit the static timing analysis (STA) model, thus reduce the frequency that the incremental static timing analysis (iSTA) tool need to be called. Han et al. [52] propose SI for Free, a regression method to predict expensive signal integrity (SI) mode sign-off timing results by using cheap non-SI mode sign-off timing analysis. [68] propose golden timer extension (GTX), a framework to reduce mismatches between different sign-off timing analysis tools to obtain neither optimistic nor pessimistic results.

Lu et al. [102] employ GAN and RL for clock tree prediction. Flip flop distribution, clock net distribution, and trial routing results serve as input images. For feature extraction, GAN-CTS adopts transfer learning from a pre-trained ResNet-50 on the ImageNet dataset by adding fully-connected (FC) layers. A conditional GAN is utilized to optimize the clock tree synthesis, of which the generator is supervised by the regression model. An RL-based policy gradient algorithm is leveraged for the clock tree synthesis optimization.

4.2.3 Placement Decision Making. As the preliminary step of the placement, floorplanning aims to roughly determine the geometric relationship among circuit modules and to estimate the cost of the design. He et al. [53] explore the possibility of acquiring local search heuristics through a learning mechanism. More specifically, an agent has been trained using a novel deep Q-learning algorithm to perform a walk in the search space by selecting a candidate neighbor solution at each step, while avoiding introducing too much prior human knowledge during the search. Google [113]

recently models chip placement as a sequential decision making problem and trains an RL policy to make placement decisions. During each episode, the RL agent lays the macro in order. After arranging macros, it utilizes the force-directed method for standard cell placement. GCN is adopted in this work to embed information related to macro features and the adjacency matrix of the netlist. Besides, FC layers are used to embed metadata. After the embedding of the macros, the graph and the metadata, another FC layer is applied for reward prediction. Such embedding is also fed into a deconvolution CNN model, called PolicyNet, to output the mask representing the current macro placement. The policy is optimized with RL to maximize the reward, which is the weighted average of wirelength and congestion.

4.3 Power Deliver Network Synthesis and IR Drop Predictions

Power delivery network (PDN) design is a complex iterative optimization task, which strongly influences the performance, area and cost of a chip. To reduce the design time, recent studies have paid attention to ML-based IR drop estimation, a time-consuming sub-task. Previous work usually adopts simulator-based IR analysis, which is challenged by the increasing complexity of chip design. IR drop can be divided into two categories: static and dynamic. Static IR drop is mainly caused by voltage deviation of the metal wires in the power grid, while dynamic IR drop is led by the switching behaviors and localized fluctuating currents. In IncPIRD [54], the authors employ XGBoost to conduct incremental prediction of static IR drop problem, which is to predict IR value changes caused by the modification of the floorplan. For dynamic IR drop estimation, Xie et al. [155] aim to predict the IR values of different locations and models IR drop estimation problem as a regression task. This work introduces a "maximum CNN" algorithm to solve the problem. Besides, PowerNet is designed to be transferable to new designs, while most previous studies train models for specific designs. A recent work [173] proposes an electromigration-induced IR drop analysis framework based on conditional GAN. The framework regards the time and selected electrical features as input images and outputs the voltage map. Another recent work [28] focuses on PDN synthesis in floorplan and placement stages. This paper designs a library of stitchable templates to represent the power grid in different layers. In the training phase, SA is adopted to choose a template. In the inference phase, MLP and CNN are used to choose the template for floorplan and placement stages, respectively. Cao et al. [16] use hybrid surrogate modeling (HSM) that combines SVM, ANN and MARS to predict the bump inductance that represents the quality of the power delivery network.

4.4 Design Challenges for 3D Integration

3D integration is gaining more attention as a promising approach to further improve the integration density. It has been widely applied in memory fabrication by stacking memory over logic.

Different from the 2D design, 3D integration introduces die-to-die variation, which does not exist in 2D modeling. The data or clock path may cross different dies in through-silicon via (TSV)-based 3D IC. Therefore, the conventional variation modeling methods, such as on-chip variation (OCV), advanced OCV (AOCV), parametric OCV (POCV), are not able to accurately capture the path delay [131]. Samal et al. [131] use MARS to model the path delay variation in 3D ICs.

3D integration also brings challenges to the design optimization due to the expanded design space and the overhead of design evaluation. To tackle these challenges, several studies [31, 122, 131] have utilized design space exploration methods based on machine learning to facilitate 3D integration optimization.

The state-of-the-art 3D placement methods [75, 121] perform bin-based tier partitioning on 2D placement and routing design. However, the bin-based partitioning can cause significant quality degradation to the 3D design because of the unawareness of the design hierarchy and technology.

Table 3. Summary of ML for logic synthesis and physical design

Section	Task	ML Algorithm	Reference
Logic Synthesis	To decide which optimizer (AIG/MIG) should be utilized for different circuits.	DNN	[115]
	To classify the optimal synthesis flows.	CNN	[167]
	To generate the optimal synthesis flows.	GCN,RL	[48]
	To generate the optimal synthesis flows.	RL	[56]
Placement	To train, predict, and evaluate potential datapaths.	SVM,NN	[150]
	To make placement decisions.	GCN,RL	[113]
Routing	To detect DRC hotspot and DRV count.	CNN	[154]
	To predict routing congestion.	CNN	[89]
		GAN	[6]
		ML	[106]
		MARS	[125]
	To predict routability of a given placement.	MARS,SVM	[18]
	To model on-chip router performance.	MARS	[63]
	To predict wirelength.	LDA, KNN	[60]
	To predict the circuit performance after placement stage.	ML	[27]
	To predict detailed routing result after global routing.	ML	[82]
	To model sign-off timing analysis.	RF	[8]
		LR	[67]
	To predict and optimize the clock tree.	GCN,CNN,RL	[102]
Power Deliver Network Synthesis and IR Drop Predictions	To predict incremental static IR drop.	XGBoost	[54]
	To predict dynamic IR drop by regressing.	CNN	[155]
	To predict electromigration-induced IR drop.	GAN	[173]
	To choose the power grid template.	MLP,CNN	[28]
	To predict bump inductance.	SVM,ANN,MARS	[16]
3D Integration	To advance the tier partition.	GNN	[103]
	To model the path delay variation.	MARS	[131]
	To optimize 3D designs.	Local Search	[31]
		BO	[122]
Other	To predict the embedded memory timing failure.	ML	[17]
	To predict aging effect.	RF	[11]

Considering the graph-like nature of the VLSI circuits, Lu et al. [103] proposed a GNN-based unsupervised framework (TP-GNN) for tier partitioning. TP-GNN first performs the hierarchy-aware edge contraction to acquire the clique-based graph where nodes within the same hierarchy can be contracted into supernodes. Moreover, the hierarchy and the timing information is included in the initial feature of each node before GNN training. Then the unsupervised GNN learning

can be applied to general 3D design. After the GNN training, the weighted k-means clustering is performed on the clique-based graph for the tier assignment based on the learned representation. The proposed TP-GNN framework is validated on experiments of RISC-V based multi-core system and NETCARD from ISPD 2012 benchmark. The experiment results indicate 7.7% better wirelength, 27.4% higher effective frequency and 20.3% performance improvement.

4.5 Other Predictions

For other parameters, Chan et al. [17] adopt HSM to predict the embedded memory timing failure during initial floorplan design. Bian et al. [11] work on aging effect prediction for high-dimensional correlated on-chip variations using random forest.

4.6 Summary of Machine Learning for Logic Synthesis and Physical Design

We summarize recent studies on ML for logic synthesis and physical design in Table 3. For logic synthesis, researchers focus on predicting and evaluating the optimal synthesis flows. Currently, these studies optimize the synthesis flow based on the primitives of existing tools. In the future, we expect to see more advanced algorithms for logic synthesis be explored, and more metrics can be formulated to evaluate the results of logic synthesis. Besides, applying machine learning to logic synthesis for emerging technologies is also an interesting direction.

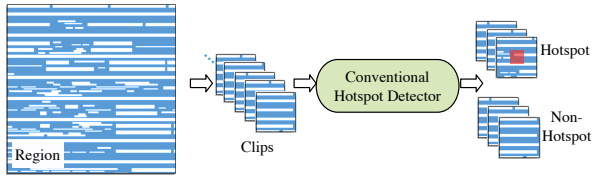
In the physical design stage, recent studies mainly aim to improve the efficiency and accuracy by predicting the related information that traditionally needs further simulation. A popular practice is to formulate the EDA task as a computer vision (CV) task. In the future, we expect to see more studies that incorporate advanced techniques (e.g., neural architecture search, automatic feature generation, unsupervised learning) to achieve better routing and placement results.

5 LITHOGRAPHY AND MASK SYNTHESIS

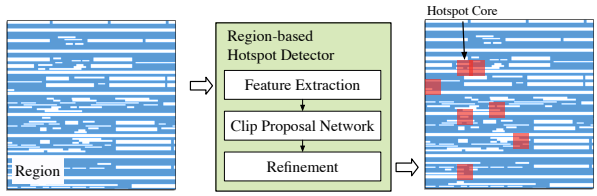
Lithography is a key step in semiconductor manufacturing, which turns the designed circuit and layout into real objects. Two popular research directions are lithography hotspot detection and mask optimization. To improve yield, lithography hotspot detection is introduced after the physical implementation flow to identify process-sensitive patterns prior to the manufacturing. The complete optical simulation is always time-consuming, so it is necessary to analyze the routed layout by machine learning to reduce lithography hotspots in the early stages. Mask optimization tries to compensate diffraction information loss of design patterns such that the remaining pattern after lithography is as close to the design patterns as possible. Mask optimization plays an important role in VLSI design and fabrication flow, which is a very complicated optimization problem with high verification costs caused by expensive lithography simulation. Unlike the hotspot detection studies in Section 5.1 that take placement & routing stages into consideration, mask optimization focuses only on the lithography process, ensuring that the fabricated chip matches the designed layout. Optical proximity correction (OPC) and sub-resolution assist feature (SRAF) insertion are two main methods to optimize the mask and improve the printability of the target pattern.

5.1 Lithography Hotspot Detection

For lithography hotspot detection, Ding et al. [32] uses SVM for hotspot detection and small neural network for routing path prediction on each grid. To achieve better feature representation, Yang et al. [162] introduces feature tensor extraction, which is aware of the spatial relations of layout patterns. This work develops a batch-biased learning algorithm, which provides better trade-offs between accuracy and false alarms. Besides, there are also attempts to check inter-layer failures with deep learning solutions. A representative solution is proposed by Yang et al. [161]. They employ an adaptive squish layout representation for efficient metal-to-via failure check. Different



(a) Traditional Hotspot Detection



(b) Region-based Hotspot Detection

Fig. 6. Region-based hotspot detection promises better performance (reproduced from [22]).

layout-friendly neural network architectures are also investigated these include vanilla VGG [160], shallow CNN [162] and binary ResNet [65].

With the increased chip complexity, traditional deep learning/machine learning-based solutions are facing challenges from both runtime and detection accuracy. Chen et al. [22] recently propose an end-to-end trainable object detection model for large scale hotspot detection. The framework takes the input of a full/large-scale layout design and localizes the area that hotspots might occur (see Figure 6). In [44], an attention-based CNN with inception-based backbone is developed for better feature embeddings.

5.2 Machine Learning for Optical Proximity Correction

For OPC, inverse lithography technique (ILT) and model-based OPC are two representative mask optimization methodologies, and each of which has its own advantages and disadvantages. Yang et al. [163] propose a heterogeneous OPC framework that assists mask layout optimization, where a deterministic ML model is built to choose the appropriate one from multiple OPC solutions for a given design, as shown in Figure 7.

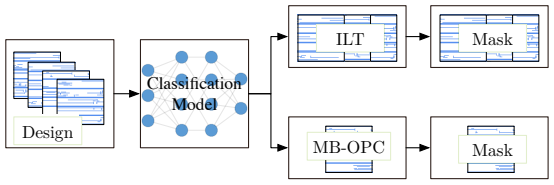


Fig. 7. A heterogeneous OPC framework (reproduced from [163]).

With the improvement of semiconductor technology and the scaling down of ICs, traditional OPC methodologies are becoming more and more complicated and time-consuming. Yang et al. [159] propose a new OPC method based on generative adversarial network (GAN). A Generator (G) is used to generate the mask pattern from the target pattern, and a discriminator (D) is used to estimate the quality of the generated mask. GAN-OPC can avoid complicated computation

in ILT-based OPC, but it faces the problem that the algorithm is hard to converge. To deal with this problem, ILT-guided pre-training is proposed. In the pre-training stage, the D network is replaced with the ILT convolution model, and only the G network is trained. After pre-training, the ILT model that has huge cost is removed, and the whole GAN is trained. The training flow of GAN-OPC and ILT-guided pre-training is shown in Figure 8. The experimental results show that the GAN-based methodology can accelerate ILT based OPC significantly and generate more accurate mask patterns.

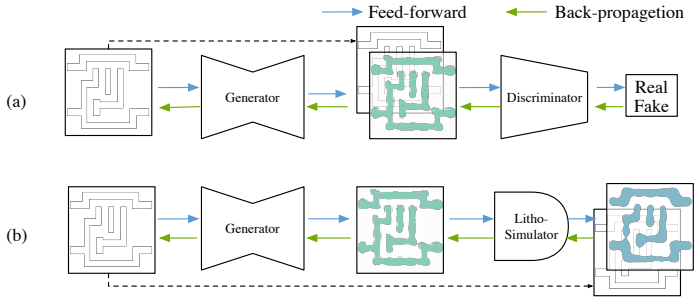


Fig. 8. The training flow of (a) GAN-OPC and (b) ILT-Guided Pre-training (reproduced from [159]).

Traditional ILT-based OPC methods are costly and result in highly complex masks where many rectangular variable-shaped-beam (VSB) shots exist. To solve this problem, Jiang et al. [64] propose an ML-based OPC algorithm named neural-ILT, which uses a neural network to replace the costly ILT process. The loss function is specially designed to reduce the mask complexity, which gives punishment to complicated output mask patterns. In addition, for fast litho-simulation, a CUDA-based accelerator is proposed as well, which can save 96% simulation time. The experimental results show that neural-ILT achieves a 70× speedup and 0.43× mask complexity compared with traditional ILT methods.

Recently, Chen et al. [20] propose DAMO, an end-to-end OPC framework to tackle the full-chip scale. The lithography simulator and mask generator share the same deep conditional GAN (DCGAN), which is dedicatedly designed and can provide a competitively high resolution. The proposed DCGAN adopts UNet++ [176] backbone and adds residual blocks at the bottleneck of UNet++. To further apply DAMO on full-chip layouts, a coarse-to-fine window splitting algorithm is proposed. First, it locates the regions of high via density and then runs KMeans++ algorithm on each cluster containing the via pattern to find the best splitting window. Results on ISPD 2019 full-chip layout show that DAMO outperforms state-of-the-art OPC solutions in both academia [43] and an industrial toolkit.

5.3 Machine Learning for SRAF Insertion

Several studies have investigated ML-aided SRAF insertion techniques. Xu et al. [158] propose an SRAF insertion framework based on ML techniques. Geng et al. [43] propose a framework with a better feature extraction strategy. Figure 9 shows the feature extraction stage. After their concentric circle area sampling (CCAS) method, high-dimension features x_t are mapped into a discriminative low-dimension features y_t through dictionary training by multiplication of an atom matrix D. The atom matrix is the dictionary consists of representative atoms of the original features. Then, the sparse codes y_t are used as the input of a machine learning model, more specifically, a logistic regression model that outputs a probability map indicating whether SRAF should be inserted at

each grid. Then, the authors formulate and solve the SRAF insertion problem as an integer linear programming based on the probability grid and various SRAF design rules.

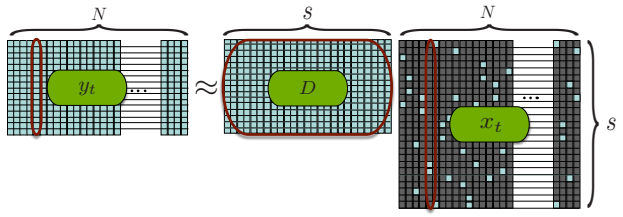


Fig. 9. Dictionary learning based feature extraction (reproduced from [43]).

5.4 Machine Learning for Lithography Simulation

There are also studies that focus on fast simulation of the tedious lithography process. Traditional lithography simulation contains multiple steps, such as optical model building, resist model building, and resist pattern generation. LithoGAN [165] proposes an end-to-end lithography modeling method by using GAN, of which the framework is shown in Figure 10. Specifically, a conditional GAN is trained to map the mask pattern to a resist pattern. However, due to the characteristic of GAN, the generated shape pattern is good, while the position of the pattern is not precise. To tackle this problem, LithoGAN adopts a conditional GAN for shape modeling and a CNN for center prediction. The experimental results show that LithoGAN can predict the resist pattern with high accuracy, and this algorithm can reduce the lithography simulation time for several orders of magnitude. [20] is also equipped with a machine learning-based lithography simulator that can output via contours accurately to assist via-oriented OPC.

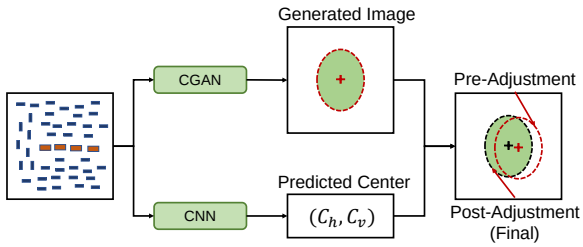


Fig. 10. The LithoGAN framework (reproduced from [165]).

5.5 Summary

This section reviews ML techniques used in the design for manufacturability stage that include lithography hotspot detection, mask optimization and lithography modeling. Related studies are summarized in Table 4.

6 ANALOG DESIGN

Despite the promotion of digital circuits, the analog counterpart is still irreplaceable in applications like nature signal processing, high speed I/O and drive electronics [126]. Unlike digital circuit design, analog design demands lots of manual work and expert knowledge, which often makes

Table 4. Summary of ML for lithography and mask optimization

Task	Work	ML Algorithm	References
Lithography Hotspot Detection	To detect single layer layout lithography hotspots.	SVM, NN	[32]
		CNN	[65, 160, 162]
	To detect multilayer layout lithography hotspots.	CNN	[161]
	To fast detect large scale lithography hotspots.	CNN	[22]
		Attention	[44]
OPC	Heterogeneous OPC	CNN	[163]
	GAN-OPC	GAN	[159]
	Neural ILT	CNN	[64]
	DAMO	DCGAN	[20]
SRAF insertion	ML-based SRAF generation	Decision Tree, Regression	[158]
	SRAF insertion	Dictionary learning	[43]
Litho-simulation	LithoGAN	CGAN, CNN	[165]
	DAMO	DCGAN	[20]

it the bottleneck of the job. For example, the analog/digital converter and Radio Frequency (RF)[1] transceiver only occupy a small fraction of area but cost the majority of design efforts in a typical mixed-signal System-on-Chip (SoC), compared to other digital processors [129].

The reason for the discrepancy can be summarized as follows: 1) Analog circuits have a larger design space in terms of device size and topology than digital circuits. Sophisticated efforts are required to achieve satisfactory results. 2) The specifications of analog design are variable for different applications. It is difficult to construct a uniform framework to evaluate and optimize different analog designs. 3) Analog signals are more susceptible to noise and process-voltage-temperature variations, which cost additional efforts in validation and verification.

6.1 The Design Flow of Analog Circuits

Gielen and Rutenbar [45] provide the design flow followed by most analog designers. As shown in Figure 11, it includes both top-down design steps from system level to device-level optimizations and bottom-up layout synthesis and verification. In the top-down flow, designers choose proper topology, which satisfies system specifications in the circuit level. Then device sizes are optimized in the device level. The topology design and device sizing constitute the pre-layout design. After the schematic is well-designed, designers draw the layout of the circuit. Then they extract parasitics from the layout and simulate the circuit with parasitics. This is known as post-layout simulations. If the post-layout simulation fails to satisfy the specifications, designers need to resize the parameters and repeat the process again. This process can go for many iterations before the layout is done [136].

Although analog design automation has improved significantly over the past few decades, automatic tools cannot replace manual work in the design flow [10] yet. Recently, researchers are trying to introduce machine learning techniques to solve analog design problems. Their attempts

[1]With a slight abuse of acronym, RF stands for both Random Forest, and Radio Frequency. The meaning should be clear from the context.

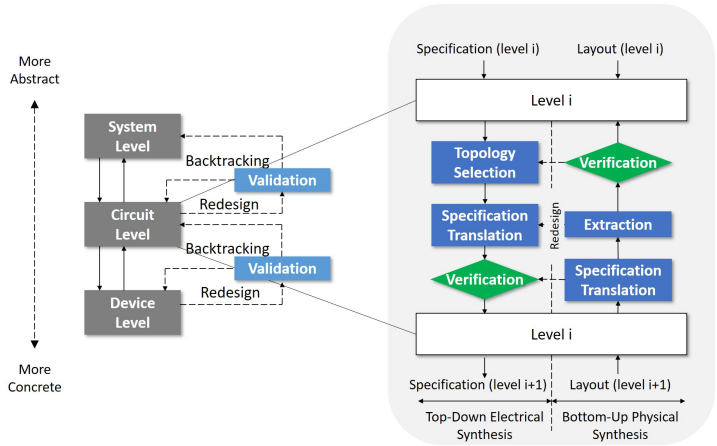


Fig. 11. Hierarchical levels of analog design flow (reproduced from [129]).

range from topology selection at the circuit level to device sizing at the device level as well as the analog layout in the physical level.

6.2 Machine Learning for Circuit Topology Design Automation

Typically, topology design is the first step of analog circuits design, followed by the determination of device sizes and parameters. The process is time-consuming, and unsuitable topology will lead to redesign from the very beginning. Traditionally, topology design relies on the knowledge and experiences of expert designers. As the scale and demand of analog circuits are increasing, CAD tools are urgently needed by engineers. Despite this, automation tools for topology design are still much less explored due to its high degree of freedom.

Researchers have attempted to use ML methods to speed up the design process. Some researchers [111, 117, 137] deal with topology selection problem, selecting the most suitable topology from several available candidate. Li et al. [83] focus on extracting well-known building blocks in circuit topology. Recently, Rotman and Wolf [130] use RNN and hypernetwork to generate two-port circuit topology.

6.2.1 Topology Selection. For common-used circuit functional units, like amplifiers, designers may not need to design from the beginning. Instead, it is possible to choose from a fixed set of available alternatives. It is a much more simple problem than designing from scratch. Early in 1996, Orzáez et al. [117], Silgado et al. [137] put forward a fuzzy-logic based topology selection tool called FASY. They use fuzzy logic to describe relationships between specifications (e.g., DC gain) and alternatives and use backpropagation to train the optimizer. More recent research [111] uses CNN as the classifier. They train CNN with circuit specifications as the inputs and the topology indexes as the labels.

The main problem with the topology selection methods is that the data collection and the training procedure are time-consuming. Therefore, topology selection is efficient only when repetitive designs are needed such that a trained model can be reused.

6.2.2 Topological Feature Extraction. One challenge of topology design automation is to make algorithms learn the complex relationships between components. To make these relationships more understandable, researchers focus on defining and extracting features from circuit topology. Li

et al. [83] present algorithms for both supervised feature extraction and unsupervised learning of new connections between known building blocks. The algorithms are also designed to find hierarchical structures, isolate generic templates (patterns), and recognize overlaps among structures. Symmetry constraint are one of the most essential topoligical features in circuits. Liu et al. [96] propose a spectral analysis method to detect system symmetry with graph similarity. With a graph representation of circuits, their method is capable of handling passive devices as well. Kunal et al. [77] propose a GNN-based methodology for automated generation of symmetry constraints. It can hierarchically detect symmetry constraints in multiple levels and works well in a variety of circuit designs.

6.2.3 Topology Generation. The aforementioned studies do not directly generate a topology. A recent study [130] makes the first attempt to generate circuit topology for given specifications. Their focus is limited to two-port circuits. They utilize an RNN and Hypernetwork to solve the topology generation problem and report better performance than the traditional methods when the inductor circuit length $n \geq 4$.

6.3 Machine Learning for Device Sizing Automation

6.3.1 Reinforcement Learning Based Device Sizing. The problem of device sizing can be formulated as follows:

$$\begin{aligned} \arg\min_{x} \quad & \sum_{x} q_c(x), \\ \text{s.t.} \quad & f_h(x) \geq y_h, \end{aligned} \tag{3}$$

where $x \in \mathbb{R}^n$ denotes the design parameters, including the size of each transistors, capacitors and resistors. $y \in \mathbb{R}^m$ denotes the specifications, including the rigid targets $y_h \in \mathbb{R}^{m_1}$ such as bandwidths, DC gains or phase margins and the optimization targets $y_o \in \mathbb{R}^{m_2}$ such as power or area. The simulator f is defined as the map from parameters to specifications. To normalize the contribution of different specifications, the objective function is defined as $q_c(x) = f_o(x)/y_o$.

Based on this optimization model, Wang et al. [148] apply the reinforcement learning technique to deal with device sizing problems. Figure 12 illustrates the proposed reinforcement learning framework. At each environment step, the observations from the simulator are fed to the agent. A reward is calculated by the value network based on current performance. Then, the agent responds with an action to update the device sizes. Because the transistors are both affected by their local status (e.g., transconductance g_m, drain current I_{ds}, etc.) and the global status (DC operating points) of the circuit, the optimization of each transistor is not independent. To promote learning performance and efficiency, the authors use a multi-step environment, where the agent receives both the local status of the corresponding transistor and the global status.

Although the device sizing problem is automated by the reinforcement learning approach, the training process depends heavily on efficient simulation tools. However, current simulation tools can only satisfy the need for schematic simulations. As for post-layout simulation that requires parasitic extraction, the time of each training iteration increases significantly. To reduce the simulation overhead, Settaluri et al. [134] introduce transfer learning techniques into reinforcement learning. In the proposed approach, the agent is trained by schematic simulations and validated by post-layout simulations. The authors show that, with some additional iterations on deployment, the proposed approach can bring 9.4× acceleration compared to previous approaches.

Following their previous work [148], the authors utilize GCN to enhance the transferability of reinforcement learning methods [147]. Unlike traditional multi-layer agents, the GCN-based agent extracts topology information from circuit netlists. In a GCN layer, each transistor is represented

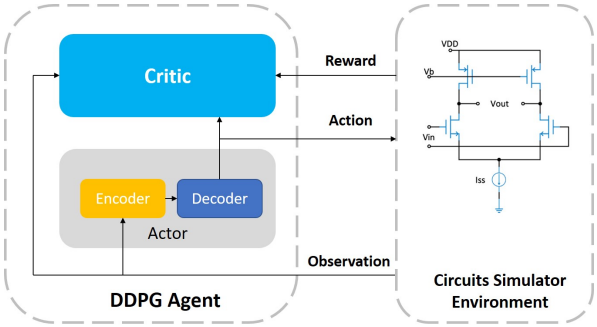


Fig. 12. The framework of reinforcement learning for device sizing (reproduced from [148]).

by a hidden neuron calculated by aggregating feature vectors from its neighbors. Specifically, the calculation can be written as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})H^{(l)}W^{(l)}, \tag{4}$$

where $\tilde{A}$ is the adjacency matrix A of the circuit topology plus the identity matrix I_N. $\tilde{D}_{ii} = \sum \tilde{A}_{ij}$ is a diagonal matrix. And $H^{(l+1)}$ is the hidden features of the lth layer. The weight matrix $W^{(l)}$ is a trainable matrix updated by Deep Deterministic Policy Gradient (DDPG) [92]. Because different circuits with the same function have similar design principles (e.g., two-stage and three-stage amplifier). The weight matrix trained for one circuit can be reused by another circuit. Besides the topologies transferring, the GCN-based RL agent is able to port existing designs from one technology node to another by sharing the weight matrices.

6.3.2 Artificial Neural Network Based Device Sizing. Rosa et al. [129] propose a data augmentation method to increase the generalization ability of the trained ML model. Specifically, the original training set T is replaced by augmented set $T' = T \cup T_1 \cup T_2 ... \cup T_k$. For each T_i, the relationship between its sample x_i' and original sample x_i is formulated as follows:

$$x_i' = x_i + (\frac{\gamma}{M}\sum_{j=1}^{M} x_j)\Delta\Gamma, \tag{5}$$

where $\gamma \in [0, 1]$ is a hyper-parameter used to adjust the mean value. And Δ and Γ denote a diagonal matrix composed by random value of [0,1] and value in $-1, 1$, respectively. For $spec_i$ to maximize like DC gain, Γ_i takes value -1. Conversely, it takes value 1 for $spec_i$ to minimize, like power or area. As a result, K copies with worse specifications are generated for each sample x_i. The models trained on the augmented dataset are more robust.

Besides the augmentation method, this paper proposes to use a 3-layer MLP model to conduct regression and classification. Given circuit performances as the input, the model outputs circuit information as two parts: 1) the size of devices in multiple topologies; 2) the classification of different topologies. The device sizing problem is solved by regression, while the topology selection is solved by classification. Compared to the simple regression models, the regression-and-classification model obtains the best performance.

6.3.3 Machine Learning Based Prediction Methods. As mentioned above, the time cost by simulation is the main overhead of training models, especially for post-layout simulation. In order to speed up the training process, Hakhamaneshi et al. [51] and Pan et al. [119] use DNN and SVM to predict the simulator. Hakhamaneshi et al. [51] use the device information of two circuits as the input of the

Table 5. Comparison of different device sizing methods

ML Algorithm	Simulation tools	Nums of simulation (Two-Stage OP)	Reference
Reinforcement learning	Commercial tools	1e4	[148]
Genetic algorithm	DNN	1e2	[51]
Reinforcement learning+Transfer learning	Commercial tools	1e4/10 (training/deployment)	[134]
Reinforcement learning+GCN	Commercial tools	1e4/100 (training/deployment)	[147]
ANN	Commercial tools	1e4/1 (training/deployment)	[129]
Genetic algorithm	SVM	1e2	[119]

DNN predictor. The model outputs the relative superiority of the two circuits on each specification instead of the absolute values. Because the prediction problem is non-convex and even ill-posed, and the training data is also limited by computational resources. Learning to compare (predict the superiority) is a relatively easy task compared to directly fitting each specification. Besides, enumerating each pair of circuit designs enlarge the training set by $\frac{N}{2}\times$, where N denotes the number of circuit designs.

6.3.4 Comparison and Discussion on Device Sizing. Table 5 lists introduced methods and their performance. The widely-studied two-stage operational amplifier (OPA) is adopted as an example for comparison. Instead of performance, sample efficiency is used as the criterion because two-stage OPA is a relatively simple design and different algorithms can achieve comparable circuit performance. It is shown that machine learning algorithms require more simulations on the training phase than traditional genetic methods. But only a few iterations of inference are needed when deploying the model. Thus, ML-based methods have more potential in large scale applications at the cost of increased training costs. On the other hand, genetic algorithms combined with ML-based predictor is a popular solution to reduce the number of needed simulations. Note that different learning algorithms have been adequately verified on simple designs like two-stage OPA. However, designing complicated circuits is still challenging.

6.4 Machine Learning for Analog Layout

Analog layout is a hard problem because the parasitics in the layout have a significant impact on circuit performances. This leads to a performance difference between pre-layout and post-layout simulations. Meanwhile, the relation between layout and performance is complex. Traditionally, circuit designers estimate parasitics according to their experience, leading to a long design time and potentials for inaccuracies [128]. Therefore, automated analog layout has drawn attention from researchers. Recently, the development of machine learning algorithms promotes research on this problem. All the studies introduced below are summarized in Table 6.

Xu et al. [156] use GAN to guide the layout generation. The network learns and mimics designers' behavior from manual layouts. Experiments show that generated wells have comparable post-layout circuit performance with manual designs on the op-amp circuit. Kunal et al. [76] train a GCN to partition circuit hierarchy. The network takes circuit netlist as input and outputs circuit hierarchy. With postprocessing, the framework reaches 100% accuracy in 275 test cases. Zhang et al. [168] introduce a GNN to estimate Electromagnetic (EM) properties of distributed circuits. And they inversely use the model to design circuits with targeted EM properties. Zhu et al. [177] propose a

Table 6. Summary of ML for analog layout

Stage	Task	ML Algorithm	References
Pre-layout Preparation	circuit hierarchy generation	GCN	[76]
	parasitics estimation	GNN	[128]
		Random Forest	[136]
Layout Generation	well generation	GAN	[156]
	closed-loop layout synthesis	Bayesian Optimization	[98]
	routing	VAE	[177]
Post-layer Evaluation	electromagnetic properties estimation	GNN	[168]
	performance prediction	SVM, random forest, NN	[85]
		CNN	[97]
		GNN	[84]

fully automated routing framework based on the variational autoencoder (VAE) algorithm. Wu et al. [152] design a knowledge-based methodology. They compare the targeted circuit with legacy designs to find the best match. Meanwhile, they expand the legacy database when new circuits are designed. Liu et al. [98] put forward a closed-loop design framework. They use a multi-objective Bayesian optimization method to explore circuit layout and use simulation results as the feedback.

To close the gap between pre-layout and post-layout simulations, some researchers attempt to estimate parasitics before layout. Ren et al. [128] use GNN to predict net parasitic capacity and device parameters based on the circuit schematic. Shook et al. [136] define several net features and use a random forest to regress net parasitic resistance and capacity. They also model the multi-port net with a star topology to simplify the circuits. Experiments show that with estimated parasitics, the error between pre-layout and post-layout circuit simulation reduces from 37% to 8% on average.

Typically, post-layout simulations with SPICE-like simulators are time-consuming. So many researchers focus on layout performance prediction with ML algorithms. Li et al. [85] compare the prediction accuracy of three classical ML algorithms: SVM, random forest, and nerual network. They also combine the performance prediction algorithms with simulated annealing to fulfill an automated layout framework. Liu et al. [97] propose a 3D CNN for circuit inputs. First, circuits are converted to 2D images. Then a third coordinate channel is added to the image to form 3D inputs. Li et al. [84] propose a customized GNN for performance prediction. They report a higher accuracy than the CNN-based method [97].

6.5 Conclusion of Analog Design

The power of machine learning algorithms has been demonstrated extensively for analog device sizing, topology design and layout problems. Compared to previous optimization-based algorithms, machine learning methods require fewer simulation rounds but achieve higher quality designs. However, existing methods cannot replace human experts yet in the analog design flow. One obstacle is that the models are learned from a limited dataset and have limited flexibility. Most researchers train and test their method on typical circuits like OTAs. A generalizable model designed for a variety of circuits is desired in the future study. Another challenge is that the vast space of

system-level design has not been studied. The potential of machine learning in analog design may be further exploited in the future.

7 VERIFICATION AND TESTING

Verification and testing of a circuit are complicated and expensive processes due to the coverage requirements and the high complexity. Verification is conducted in each stage of the EDA flow to ensure that the designed chip has correct functions. On the other hand, testing is necessary for a fabricated chip. Note that from many perspectives, verification and testing share common ideas and strategies, meanwhile face similar challenges. For instance, with the diversity of applications and the complexity of the design, traditional formal/specification verification and testing may no longer meet various demands.

For the coverage requirements, a circuit or system can be very complex and may have many different functions corresponding to different input data. To verify a system with low cost, the test set design should be compact and avoid containing "repeated" or "useless" situations with covering enough combinations of inputs to ensure reliability. Therefore, a well-selected test set and a proper strategy are crucial to the fast and correct verification. Traditionally, for test set design, random generation algorithms and Automated Test Pattern Generation (ATPG) are usually used in the verification stage and the testing stage, respectively. And their designs are always far from the optimal solution. Therefore, it is intuitive to optimize the verification process by reducing the redundancy of the test set.

High complexity of chip testing/verification is another problem. For example, in the analog/RF system design, it is expensive and time-consuming to test the performance accurately or to verify the SPICE netlist formally. Predicting accurate results with low precision test results derived from cheap testing methods is a promising solution to this problem.

To meet the coverage requirements and reduce complexity, more and more ML algorithms are applied in the verification and testing process, to make fast analog/RF system testing, build simplified estimation model, infer and predict the verification results, optimize sample strategies, and even generate high quality test benches. These methodologies can be divided into two categories: 1) Machine learning for test set redundancy reduction, which is applied in both verification and testing stage; 2) Machine learning for complexity reduction, which is applied in chip testing, verification, and diagnosis.

7.1 Machine Learning for Test Set Redundancy Reduction

Coverage is the primary concern when designing a test set in verification and testing problems. However, the definition of "coverage" is different in different problems. For example, for digital design, the test set is supposed to cover as many states of the finite state machine (FSM) or input situations as possible. For analog/RF design, since the input is continuous and the system can be very sensitive to environmental disturbance, a sampling strategy that can cover most input values and working situations is needed. As for the test of semiconductor technology, a test point is a design that needs to be synthesized or fabricated, and the test set needs to cover the whole technology library. We will introduce these problems and corresponding studies based on ML techniques.

7.1.1 Test Set Redundancy Reduction for Digital Design Verification. The verification of a digital design will be carried out in each stage of the EDA flow, in which the verification space of a digital design under test (DUT) usually includes a huge number of situations. Thus, manually designing the test set requires rich expertise and is not scalable. Originally, the test set is usually generated by a biased random test generator with some constraints [62], which can be configured by setting

a series of directives. Later on, Coverage-Directed test Generation (CDG) techniques have been explored to optimize the test set generation process. The basic idea of CDG is to simulate, monitor, and evaluate the coverage contribution of different combinations of input and initial state. And then, the derived results are used to guide the generation of the test set. There are many CDG works that are based on various ML algorithms such as Bayesian Network [38], Markov Model [145], Genetic Algorithm [49, 135], rule learning [33, 57, 69], SVM [24, 47] and NN [146]. We refer the readers to [62] for a more detailed survey on related papers before 2012. Note that although the word "test" is mentioned frequently in this field, these works mainly aim at aiding the verification process.

GA can be applied in CDG problems. Shen et al. [135] combine the biased random test generation with GA. First, a constraint model is described and encoded, then a set of constraint models with different configurations is sent into the simulator to evaluate the coverage performance. GA method is used to search for a better configuration with higher coverage. Habibi et al. [49] propose a high-level hardware modeling methodology to get a better description of FSM states and use GA to find a proper configuration of the test generator.

Beyond the traditional search strategies, more studies incorporated ML-based models to guide the search process. Chen et al. [24] use a one-class SVM for novel test detection. They assume that novel test instances are more useful and could cover more specific corners, and a one-class SVM is used to find these novel instances. Guzey et al. [47] conduct the functional test selection by using unsupervised Support Vector Analysis. The basic idea is to cluster all the input operations into several groups (e.g., AND operation, other logic operation, and all other operations). Then, one can select the relevant test subsets for specific functional verification. A recent study [146] focuses on clustering input instructions and adopts an ANN-based method to decide whether a single input instruction should be verified.

Probabilistic models are also adopted to model the DUT behavior or the test generator. Fine and Ziv [38] propose a CDG method by building a Bayesian Network between the test generator directives and coverage variables. To model the influence of input, some hidden layers are added to the network with expert domain knowledge to help explain the relationship. The Bayesian Network is dynamic and can be adjusted according to stimuli results to get a more precise model. Then, we can change the directives to achieve better coverage by running inference on the Bayesian Network. Markov model is a special case of Bayesian Network, and Wagner et al. [145] propose a Markov model for more efficient microprocessor verification. The proposed Markov model shows the transfer probability of different types of instructions. Activity monitors are used for coverage estimation, and the results are used for the adjustment of the Markov model. The learned Markov model is used as the test generator to achieve better coverage performance.

To extract more interpretable and compact knowledge from previous verification experiences, rule learning techniques also play a role in CDG problems. Katz et al. [69] apply a decision tree for rule learning of microarchitecture behaviors. Eder et al. [33] adopt the inductive logic programming method to discover instruction rules, which can be directly used as the directives for further test generation. Hsieh et al. [57] propose to discover a subgroup of states that differentiate the failing test cases from the success test cases. All these methods aim at extracting the internal rules of the verification problem. With the extracted rules, one can generate better test instances either manually or automatically.

7.1.2 Test Set Redundancy Reduction for Analog/RF Design Testing. The Analog/RF system testing can be divided into two aspects, including device-level testing and circuit-level testing. The current practice for testing an Analog/RF system is specification testing [142]. This method needs to measure the parameters of the circuit directly. The device will be continuously switched to various

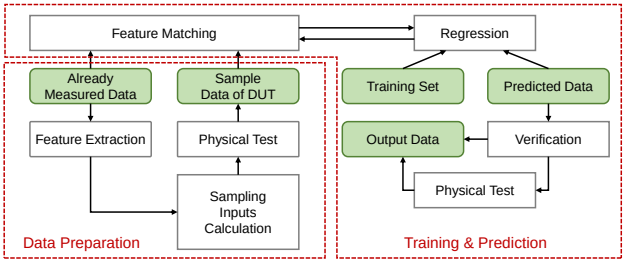


Fig. 13. The framework of [120] (reproduced from [120]).

test configurations during its operation, resulting in a long setup and establishment time. In each test configuration, measurements are performed multiple times and averaged to reduce thermal noise and crosstalk. Moreover, this complex process needs to be repeated in various modes of operation, such as temperature, voltage level, and output load. Therefore, despite the highly accurate measurement, the overall test process is extremely costly. On the other hand, specification testing requires the use of automatic test equipment (ATE), and the cost of this equipment is also very high.

A direction to solve these problems is to identify and eliminate the information redundancy in the test set by machine learning, and make a pass/fail decision only depending on a subset of it [140, 141]. In the specification test, each performance parameter may have redundant information. However, this information needs advanced statistical methods to obtain. ML can help find the complex association in the specification test, so as to reduce the types and times of the test set, and finally complete the result inference with high quality. A multi-objective genetic algorithm is applied for feature selection of the test set, which is used to extract subsets and build the prediction model based on a binary classifier to determine whether the equipment is qualified [141]. The classifier can be constructed by kNN or Ontogenic Neural Network (ONN). Taking the power set as an example, the results show that a relatively small number of non-RF specification tests (i.e., digital, DC, and low frequency) can correctly predict a large proportion of pass/fail tags. The experimental results also show that adding some RF specification tests can further improve the prediction error.

Pan et al. [120] propose a low-cost characterization method for IC technologies. They assume the devices on different dies have similar characteristics, and it is possible to use part of test samples to predict the detailed data. The framework of this work is shown in Figure 13. A small number of samples are tested, and several features are extracted from the test results. Then, the features are used to fit a regression model, with which one can infer the performance curve and predict test results of other samples. In the experiment, the authors use 267 data samples to predict 3241 data points with 0.3% average error, which reaches a 14x speedup in the test process.

7.1.3 Test Set Redundancy Reduction for Semiconductor Technology Testing. Sometimes the problem is to test a new semiconductor technology rather than a specific design. In this situation, a test instance is a synthesized or fabricated chip design, and building a test set can be extremely expensive. This problem is a little different from the testing problems mentioned before, but the idea of reducing the test set redundancy is still working. If we can predict the test set quality and select good parts in advance, the cost can be reduced significantly. Liu et al. [100] focus on optimizing the test set design via ML, of which the proposed flow is shown in Figure 14. In a traditional testing flow, every possible configuration in a logic library is synthesized, which causes huge time and energy consumption. To alleviate the problem, this work uses RF models to predict whether a test datum is

"unique" and "testable" with several features (e.g., the number of nets, fanout, and max logic depth). "Unique" means that the test data has a different logical structure compared to other test data, and "testable" means that this test data can cover a great number of IP fault. The experimental results show that this work can achieve over 11× synthesis time reduction.

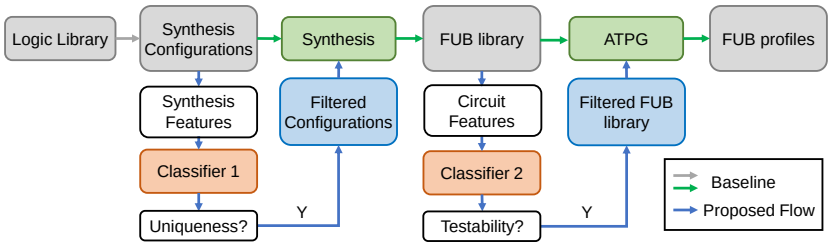


Fig. 14. The flow of proposed method in [100] (reproduced from [100]).

7.2 Machine Learning for Test & Diagnosis Complexity Reduction

7.2.1 Test Complexity Reduction for Digital Design. Recently, GCNs are used to solve the observation point insertion problem for the testing stage [104]. Inserting an observation point between the output of module 1 and the input of module 2 will make the test results of module 1 observable and the test inputs of module 2 controllable. Ma et al. [104] propose to use GCN to insert fewer test observation points while maximizing the fault coverage. More specifically, the netlist is first mapped to a directed graph, in which nodes represent modules, and edges represent wires. Then, the nodes are labeled as easy-to-observe or difficult-to-observe, and a GCN classifier is trained. Compared with commercial test tools, this method based on GCN can reduce the observation points by 11% under similar fault coverage, and reduce the test pattern count by 6%. Note that compared with other studies discussed before, observation point insertion reduces the test complexity in a different way, by decoupling the test of different modules.

7.2.2 Verification Diagnosis Complexity Reduction for Digital Design. During the verification process, a complicated diagnosis is needed whenever a bug is detected. However, this diagnosis process might be redundant sometimes since there are lots of similar bugs caused by the same hardware problem, and one situation can be analyzed repeatedly. To alleviate this problem, Mammo et al. [110] propose an automatic hardware diagnosis method named BugMD, which can classify different bugs and localize their corresponding module. With this framework, the emerging bugs can be analyzed without a complicated diagnosis process. First, the instruction windows containing bugs are encoded to input feature vectors based on the mismatch between DUT and a golden instruction set simulator, then the feature vectors are sent to a classifier for further triaging and localizing, where the ML algorithm can be a decision tree, RF, SVM or NN. To produce sufficient training data, a synthetic bug injection framework is proposed, which is realized by randomly change the functionality of several modules. The experimental results prove the feasibility of BugMD with over 90% top-3 localization accuracy.

7.2.3 Verification & Test Complexity Reduction for Analog/RF Design. With increasing system complexity and rising demand for robustness, Analog/RF signal verification has become a key bottleneck [19], which makes failure detection and design verification very challenging.

A feasible way to reduce the cost of Analog/RF system verification is to use low-cost test equipment to obtain simple results. Then ML models can be used to map from simple results to

complex results obtained by specification testing [5, 34]. The basic assumption is that the training set reflects the statistical mechanisms of the manufacturing process, thus the learned mapping can generalize for new device instances. Nevertheless, the ML model might fail to capture the correct mapping for some devices since the actual mapping is complex and is not a one-to-one mapping. Thus, a two-tier test method combining machine learning and specification testing is proposed to improve the accuracy of results [142]. During the process, the equipment is first tested by low-cost machine learning-based testing, and the reliability of the results is evaluated. If it is considered insufficient, the more expensive specification testing is conducted. An Ontogenic Neural Network (ONN) is designed to identify the ambiguous regions, and forward the devices to the specification testing. This two-tier approach achieves a trade-off between accuracy and cost.

Although formal verifications can provide guarantees for the specifications under check, they are only feasible for small analog blocks with idealistic models and fail for practical usage on large detailed SPICE circuit netlist. Therefore, machine learning is applied to aid the verification process. HFMV [58] combines a machine learning model with formal verification: When there is insufficient confidence in the test results of the machine learning model, formal verification is performed. HFMV proposes a probabilistic machine learning model to check whether there is enough confidence to meet the target specification. As shown in Figure 15, HFMV relies on two active learning approaches to improve the performance of the ML model, including 1) max variance learning to reduce model uncertainty; 2) formally-guided active learning to discover rare failure regions. Their results show that HFMV can detect rare failures.

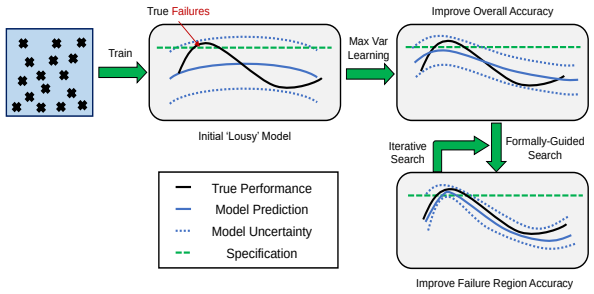


Fig. 15. Active learning for circuits testing (reproduced from [58]).

7.3 Summary of ML for Verification and Testing

There are mainly two ways of accelerating the verification and testing process: 1) Reducing the test set redundancy; 2) Reducing the complexity of the testing, verification and diagnosis process. To reduce the test set redundancy or to optimize the generation of test instances, coverage-directed test generation has been studied for a long time, which can be aided by lots of ML algorithms. Recently, test set redundancy reduction of analog/RF design or even the test of semiconductor technology have raised a lot of attention, and more ML methods are applied to solve these problems. As for reducing the verification & test complexity, there are studies that adopt low-cost tests for analog/RF design, and some other studies that focus on fast bug classification and localization. The related works on ML for verification & testing problems are summarized in Table 7.

Table 7. Summary of ML for verification and testing

Optimization Idea	Task	ML Algorithm	References
Test Set Redundancy Reduction	Digital Design	Statistical Model	[38], [145]
		Search Methods	[135], [49]
		Rule Learning	[57], [69], [33]
		CNN, SVM, et al.	[24], [47], [146]
		GCN	[104]
	Analog/RF Design	KNN, ONN	[141]
		Regression	[120]
	Semiconductor Technology	CNN	[100]
Test Complexity Reduction	Digital Design	SVM, MLP, CNN, et al.	[110]
	Analog/RF Design	ONN	[142]
		Active Learning	[58]

8 OTHER RELATED STUDIES

8.1 Power Prediction

Power estimation is necessary in electronic system design, which can be carried out at different levels according to application scenarios. In general, there is a tradeoff between the power estimation accuracy and simulation method complexity. For example, the gate-level estimation can generate a cycle-by-cycle power track with high accuracy but has a huge time consumption. In contrast, high-level simulation can only provide less accurate evaluation, but requires less specific knowledge and computing complexity at the same time. Nevertheless, it is possible for ML methods to make accurate and detailed power prediction only with high-level evaluation, which shows significant benefits for fast chip design and verification.

Lee and Gerstlauer [81] propose a multi-level power modeling method, which only uses high-level C/C++ behavior description and some hardware information to obtain power model at different granularities. The derived power model granularity depends on how much information we have about the hardware design, i.e., black, grey, or white box modeling. For each modeling problem, an evaluation flow is designed, and several regression algorithms are applied. The proposed flow achieves a significant speedup compared with traditional RTL-level or gate-level simulation within 10% error.

Kim et al. [72] propose an RTL-level power prediction framework named SIMMANI with signal clustering and power model regression. All the signals are encoded according to the toggle patterns observed in a specific window, which are then clustered and selected. The regression model takes the selected signals as the input, and outputs the power estimation result.

Besides traditional regression methods, other ML methods also show great potential in power predicting problems. PRIMAL [175] is an RTL power estimation framework based on several ML methods, including Principal Component Analysis (PCA), MLP and CNN. In PRIMAL, the toggle patterns of registers are first encoded into 1D or 2D features and then processed by various ML algorithms. The trained model can evaluate the power track for new workloads that are very different from the training set. To enhance the local information, a graph-based partitioning method

is leveraged for the mapping strategy from registers to feature pixels. PRIMAL can achieve a 50× speedup than gate-level power estimation flow with an average error below 5%.

PRIMAL is a promising solution to RTL power prediction. However, there exist transferability problems with this solution in that a power model can only describe a specific system design. That is to say, we have to train a new model for a new design. To solve this problem, a GNN-based framework named GRANNITE is proposed by Zhang et al. [169]. Different from PRIMAL, GRANNITE takes the gate-level netlist into consideration to build a GNN. GRANNITE shows good transferability among different designs by utilizing more hardware details. Note that this work still conducts an RTL-level power prediction, since the gate-level netlist is only used for the graph generation, and no gate-level power estimation is involved. Compared to a traditional probabilistic switching activity estimation, GRANNITE achieves a speed up of two orders of magnitude on average, and the average relative error is within 5.5%.

8.2 Machine Learning for SAT Solver

SAT plays an important role in circuit design and verification, error diagnosis, model detection of finite state machines, FPGA routing, logic synthesis and mapping, register allocation, timing, etc. Researchers contribute to improving the efficiency of the search engine in SAT solvers and design various strategies and heuristics. Recently, with the advancement of NNs in representation learning and solving optimization problems, there have been increasing interests in generating and solving SAT formula with NNs.

The performance of the conflict-driven Davis Putnam style SAT solver largely depends on the quality of restart strategies. Haim and Walsh [50] successfully apply a supervised learning method to design LMPick, a restart strategy selector. Among various heuristics, branching heuristics [40, 46, 88, 114] attract lots of attention for its great performance. Multi-class SVM is applied in [139] to tune parameters of heuristics, according to the features of both input and output clauses. SATzilla [157] integrates several solvers and builds an empirical hardness model for solver selection. Some work [42, 61, 71] evolve heuristics through genetic algorithms by combining existing primitives, with the latter two aiming at specializing the created heuristics to particular problem classes. There have also been other approaches utilizing reinforcement learning to discover variable selection heuristics [41, 79, 86–88].

Recently, NNs have found their applications in solving SAT. Palm et al. [118] introduce the recurrent relational network to solve relational inference, e.g. Sudoku. Evans et al. [35] present an NN architecture that can learn to predict whether one propositional formula entails another by randomly sampling and evaluating candidate assignments. There have also been several recent papers showing that various neural network architectures can learn good heuristics for NP-hard combinatorial optimization problems [9, 70, 144]. Selsam et al. [133] propose to train a GNN (called NeuroSAT) to classify SAT problems as satisfiable or unsatisfiable. Selsam and Bjørner [132] also use a simplified NeuroSAT to guide the search process of an existing solver.

In recent studies, a common practice is to use GNN for feature extraction and reinforcement learning for learning the policy. Lederman et al. [80] learn improved heuristics to solve quantified Boolean formulas via reinforcement learning while using GNN for formula encoding. Yolcu and Póczos [166] also use RL to learn local search heuristics with a GNN serving as the policy network for variable selection. Besides GNN, RNN can also be employed for formula or DAG embedding. Lately, Amizadeh et al. [7] propose Circuit-SAT to solve SAT problems, employing gated recurrent units that can implement sequential propagation of DAG-structured data. The training procedure works in the exploration and exploitation manner, which is similar to the reinforcement learning paradigm.

8.3 Acceleration with Deep Learning Engine

EDA tools typically involve solving large-scale optimization problems with heavy numerical computation, especially at the physical design stage, and extensive work is devoted to accelerating these solvers with modern parallel computing hardware like multicore CPUs or GPUs [26, 29, 101]. Many recent studies have explored GPU's opportunity in EDA problems [59, 93, 170, 171]. Still, developing good GPU implementation of EDA algorithms is challenging.

Lin et al. [93] leverage the mature deep learning engines to build a GPU-accelerated placement framework called DREAMPlace Advancement in ML has encouraged the development of software frameworks and tool-kits which decouple algorithmic description from system implementation (e.g., interaction with GPUs, optimizing low-level operator code) to help develop ML models productively [3, 123]. The key insight of this paper is that the analytical placement problem is analogous to the training of a NN model. They both involve optimizing some parameters (i.e., cell locations in placement, weights in NN) to minimize a cost function (i.e., wirelength in placement, cross-entropy loss in NN). With hand-optimized key operators integrated in DL training framework PyTorch, DREAMPlace demonstrates over $40\times$ speedup against CPU-based multi-threaded tools [26, 101]. The tool claims to be extensible to new solvers by simply adding algorithmic description in high-level languages like Python.

8.4 Auto-tuning design flow

With the increasing complexity of chip design, massive choices and parameters of the synthesis tools make up huge design space. To improve the efficiency of tuning, recent studies employ more advanced learning-based algorithms. In [179], some complete parameter settings are selected and then gradually adapted during synthesis to achieve optimal results. Kwon et al. [78] propose the first recommender system based on the collaborative filtering algorithm. The system consists of two modules: the offline learning module and the online recommendation module. The offline learning module is to predict QoR given macro specification, parameter configuration, cost function and iterative synthesis output. The online recommendation module generates several optimal settings. A recent study [153] also employs a tree-based XGBoost model for efficient tuning. Besides, this paper also designs a clustering technique that leverages prior knowledge and an approximate sampling strategy to balance exploration and exploitation. In [4], a deep RL framework that adopts unsupervised GNN to generate features is developed to automatically tune the placement tool parameters.

9 DISCUSSION FROM THE MACHINE LEARNING PERSPECTIVE

In this section, we revisit some aforementioned research studies from an ML-application perspective.

9.1 The Functionality of ML

Section 2.2 introduces the major ML models and algorithms used in EDA problems. Based on the functionality of ML in the EDA workflow, we can group most researches into four categories: decision making in traditional methods, performance prediction, black-box optimization, and automated design.

Decision making in traditional methods. The configurations of EDA tools, including the choice of algorithm or hyper-parameters, have a strong impact on the efficiency of the procedure and quality of the outcome. This class of researches utilizes ML models to replace brute-force or empirical methods when deciding configurations. ML has been used to select among available tool-chains for logic synthesis [115, 167] , mask synthesis [163], and topology selection in analog

Table 8. Overview of ML functionality in EDA tasks

ML Functionality	Task / Design Stage	ML Algorithm	Input	Output	Section
Decision making in traditional methods	HLS Design space exploration	Decision Tree, quadratic regression, etc.	Hardware directives (pragmas) in HLS design	Quality of hyper-parameters, e.g., initial state, termination conditions	Section 3.2.2
	Logic synthesis	DNN	RTL descriptions	Choice of the workflow and optimizer	Section 4.1
	Mask synthesis	CNN	Layout images	Choice of optimization methods	[163] in Section 5.1
Performance prediction	Analog topology design	CNN, Fuzzy logic, etc.	Analog specifications	Best topology selection	Section 6.2.1
	HLS	Linear Regression, SVM, Random Forest, XGBoost, etc.	HLS Report, workload characteristics, hardware characteristics	Resource usage, timing, etc.	Section 3.1
	Placement and routing	SVM, CNN, GAN, MARS, Random Forest etc.	Features from netlist or layout image	Wire-length, routing congestion, etc.	Section 4.2
	Physical implementation (lithography hotspot detection, IR drop prediction, power estimation, etc.)	SVM, CNN, XGBoost, GAN, etc.	RTL and gate-level descriptions, technology libraries, physical implementation configurations	Existence of lithography hotspots, IR drop, path delay variation, etc	Section 5.1–4.5, 5.4, 8.1
	Verification	KNN, Ontogenic Neural Network (ONN), GCN, rule learning, SVM, CNN	Subset of test specifications or low-cost specifications	boolean pass/fail prediction	Section 7
	Device sizing	ANN	Device parameter	Possibility of constraint satisfaction	Section 6.3
ML Functionality	**Task / Design Stage**	**ML Algorithm**	**Tuning parameters**	**Optimization Objective**	**References**
Black-box optimization	HLS Design Space Exploration	Random Forest, Gaussian Process, Ensemble models, etc.	Hardware directives (pragmas) in HLS design	Quality-of-Results, including latency, area, etc.	Section 3.2.1
	3D Integration	Gaussian Process, Neural Network	Physical design configurations	Clock skew, thermal performance, etc.	Section 4.4
Automated design	Logic synthesis	RL, GCN	Gate-level DAG for a logic function	Area, latency, etc.	Section 4.1
	Placement	RL, GCN	Macro placement position	Wire-length, congestion, etc.	[113] in Section 4.2.3
	Mask synthesis	GAN, CNN, Decision Tree, dictionary learning, etc.	RTL and gate-level description, layout images	Generated optical proximity correction (OPC) and sub-resolution assist feature (SRAF)	Section 5.1–5.3
	Device sizing	RL, GCN, DNN, SVM	Device parameters	Satisfaction of design constraints	Section 6.3

design [111, 117, 137]. ML has also been exploited to select hyper-parameters for non-ML algorithms such as Simulated Annealing, Genetic Algorithm, etc. (refer to Section 3.2.2).

Performance prediction. This type of tasks mainly use supervised or unsupervised learning algorithms. Classification, regression and generative models are trained by former cases in real

production to estimate QoR rapidly, to assist engineers to drop unqualified designs without time-consuming simulation or synthesis.

ML-based performance prediction is a very common type of ML application. Typical applications of this type include congestion prediction in placement & routing and hotspot detection in manufacturability estimation (Table 8). The most commonly-used models are Linear Regression, Random Forests, XGBoost, and prevailing CNNs.

Black-box optimization. This type of tasks mainly use active learning. Many tasks in EDA are DSE, i.e., searching for an optimal (single- or multi-objective) design point in a design space. Leveraging ML in these problems usually yields black-box optimization, which means that the search for optimum is guided by a surrogate ML model, not an explicit analytical model or hill-climbing techniques. The ML model learns from previously-explored design points and guides the search direction by making predictions on new design points. Different from the first category, the ML model is trained in an active-learning process rather than on a static dataset, and the inputs are usually a set of configurable parameters rather than results from other design stages.

Black-box optimization is widely used for DSE in many EDA problems. Related ML theories and how to combine with the EDA domain knowledge are extensively studied in literature. Typical applications of this type include tuning HLS-level parameters and physical parameters of 3D integration (see Table 8). The key techniques are to find an underlying surrogate model and a search strategy to sample new design points. Options of the surrogate model include GP, along with all the models used in performance prediction [105, 112]. Search strategies are usually heuristics from domain knowledge, including uniformly random exploration [95], exploring the most uncertain designs [180], exploring and eliminating the worst designs [112], etc.

Automated design. Some studies leverage AI to automate design tasks that rely heavily on human efforts. Typical applications are placement [113] and analog device sizing [134, 147, 148]. At first look it is similar to black-box optimization, but we highlight the differences as:

- The design space can be larger and more complex, for example in placement, the locations of all the cells.
- Instead of searching in the decision space, there exists **a trainable decision-making policy** that outputs the decisions, which is usually learned with RL techniques.

More complicated algorithms with large volumes of parameters, such as deep reinforcement learning, are used in these problems. This stream of researches show the potential to fully automate IC design.

Table 8 summarizes representative work of each category and typical model settings in terms of algorithm, input and output.

9.2 Data Preparation

The volume and quality of the dataset are essential to model performance. Almost all studies we review make some discussions on leveraging EDA domain knowledge to engineer a large, fair and clean dataset.

Raw data collection. Raw features and ground truth / labels are two types of data needed by ML models. Raw feature extraction is often a problem-specific design, but there are some shared heuristics. Some studies treat the layout as images and leverage image processing algorithms [32, 89, 154]. Some choose geometric or graph-based features from the netlist [150]. Some use traditional algorithms to generate features [6, 67, 106, 154]. Quite a lot studies choose features manually [6, 11, 16, 17, 27, 82, 115]. To some extend, manual feature selection lacks a theoretical guarantee or practical guidance for other problems. The labels or ground truth are acquired through time-consuming simulation or synthesis. This also drives researchers to improve data efficiency by carefully architect

their models and preprocess input features, or use semi-supervised techniques [25] to expand the dataset.

Feature preprocessing. Standard practices like feature normalization and edge data removal are commonly used in the preprocessing stage. Some studies also use dimension reduction techniques like PCA and LDA to further adjust input features [60].

9.3 Domain Transfer

There have been consistent efforts to make ML-based solutions more adaptive to domain shift, so as to save training from scratch for every new task. Some researches propose ML models that take specifications of the new application domain and predict results in new domain based on results acquired in original domain. This idea is used in cross-platform performance estimation of FPGA design instances [109, 116]. It would be more exciting to train AI agents to adapt to new task without preliminary information of the new domain, and recent studies show that Reinforcement Learning (RL) might be a promising approach. RL models pre-trained on one task is able to perform nicely on new tasks after a fine-tune training on the new domain [113, 134, 147], which costs much less time than training from scratch and sometimes lead to even better results.

10 CONCLUSION AND FUTURE WORK

It is promising to apply machine learning techniques in accelerating EDA tasks. In this way, the EDA tools can learn from previous experiences and solve the problem at hand more efficiently. So far machine learning techniques have found their applications in almost all stages of the EDA hierarchy. In this paper, we have provided a comprehensive review of the literature from both the EDA and the ML perspectives.

Although remarkable progress has been made in the field, we are looking forward to more studies on applying ML for EDA tasks from the following aspects.

- *Towards full-fledged ML-powered EDA tools.* In many tasks (e.g., analog/RF testing, physical design), the performance of purely using machine learning models is still difficult to meet the industrial needs. Therefore, smart combination of machine learning and the traditional method is of great importance. Current machine learning aided EDA methods may be still restricted to less flexible design spaces, or aim at solving a simplified problem. New models and algorithms are desired to be developed to make the ML models more useful in real applications.
- *Application of new ML techniques.* Very recently, some new machine learning models and methodologies (e.g., point cloud and GCN) and machine learning techniques (e.g., domain adaptation and reinforcement learning) begin to find their application in the EDA field. We expect to see a broader application of these techniques in the near future.
- *Trusted Machine Learning*. While ML holds the promise of delivering valuable insights and knowledge into the EDA flow, broad adoption of ML will rely heavily on the ability to trust their predictions/outputs. For instance, our trust in technology is based on our understanding of how it works and our assessment of its safety and reliability. To trust a decision made by an algorithm or a machine learning model, circuit designers or EDA tool users need to know that it is reliable and fair, and that it will cause no harm. We expect to see more research along this line making our automatic tool trusted.

ACKNOWLEDGMENT

This work was partly supported by National Natural Science Foundation of China (No. U19B2019, 61832007, 61621091), and the Research Grants Council of Hong Kong SAR (No. CUHK14209420).

REFERENCES

[1] 2017. Intel HLS Compiler. https://www.altera.com/.

[2] 2017. Xilinx Vivado HLS. https://www.xilinx.com/.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 265–283.

[4] Anthony Agnesina, Kyungwook Chang, and Sung Kyu Lim. 2020. VLSI Placement Parameter Optimization using Deep Reinforcement Learning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–9.

[5] Selim Sermet Akbay and Abhijit Chatterjee. 2005. Built-In Test of RF Components Using Mapped Feature Extraction Sensors. In *IEEE VLSI Test Symposium (VTS)*. 243–248.

[6] Mohamed Baker Alawieh, Wuxi Li, Yibo Lin, Love Singhal, Mahesh A. Iyer, and David Z. Pan. 2020. High-Definition Routing Congestion Prediction for Large-Scale FPGAs. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 26–31.

[7] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. 2019. Learning To Solve Circuit-SAT: An Unsupervised Differentiable Approach. In *International Conference on Learning Representations (ICLR)*.

[8] Erick Carvajal Barboza, Nishchal Shukla, Yiran Chen, and Jiang Hu. 2019. Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism. In *ACM/IEEE Design Automation Conference (DAC)*. 106.

[9] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

[10] E. Berkcan and F. Yassa. 1990. Towards Mixed Analog/Digital Design Automation: a Review. In *IEEE International Symposium on Circuits and Systems (ISCAS)*.

[11] Song Bian, Michihiro Shintani, Masayuki Hiromoto, and Takashi Sato. 2017. LSTA: Learning-Based Static Timing Analysis for High-Dimensional Correlated On-Chip Variations. In *ACM/IEEE Design Automation Conference (DAC)*. 66:1–66:6.

[12] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. 1992. A Training Algorithm for Optimal Margin Classifiers. In *Conference on Learning Theory*. 144–152.

[13] Justin A. Boyan and Andrew W. Moore. 2000. Learning Evaluation Functions to Improve Optimization by Local Search. *Journal of Machine Learning Research (JMLR)* 1 (2000), 77–112.

[14] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *International Conference on Computer-Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 6174)*. 24–40.

[15] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz S. Czajkowski, Stephen Dean Brown, and Jason Helge Anderson. 2013. LegUp: An Open-Source High-level Synthesis Tool for FPGA-based Processor/Accelerator Systems. *ACM Transactions on Embedded Computing (TECS)* 13, 2 (2013), 24:1–24:27.

[16] Yi Cao, Andrew B. Kahng, Joseph Li, Abinash Roy, Vaishnav Srinivas, and Bangqi Xu. 2019. Learning-Based Prediction of Package Power Delivery Network Quality. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 160–166.

[17] Wei-Ting Jonas Chan, Kun Young Chung, Andrew B. Kahng, Nancy D. MacDonald, and Siddhartha Nath. 2016. Learning-Based Prediction of Embedded Memory Timing Failures During Initial Floorplan Design. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 178–185.

[18] Wei-Ting Jonas Chan, Yang Du, Andrew B. Kahng, Siddhartha Nath, and Kambiz Samadi. 2016. BEOL Stack-Aware Routability Prediction from Placement Using Data Mining Techniques. In *IEEE International Conference on Computer Design (ICCD)*. 41–48.

[19] Henry Chang and Kenneth S. Kundert. 2007. Verification of Complex Analog and RF IC Designs. *Proc. IEEE* 95, 3 (2007), 622–639.

[20] Guojin Chen, Wanli Chen, Yuzhe Ma, Haoyu Yang, and Bei Yu. 2020. DAMO: Deep Agile Mask Optimization for Full Chip Scale. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[21] Jingsong Chen, Jian Kuang, Guowei Zhao, Dennis J-H Huang, and Evangeline FY Young. 2020. PROS: A Plug-in for Routability Optimization Applied in the State-of-the-art Commercial EDA Tool Using Deep Learning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[22] Ran Chen, Wei Zhong, Haoyu Yang, Hao Geng, Xuan Zeng, and Bei Yu. 2019. Faster Region-based Hotspot Detection. In *ACM/IEEE Design Automation Conference (DAC)*. 146.

[23] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. 785–794.

[24] Wen Chen, Nik Sumikawa, Li-C. Wang, Jayanta Bhadra, Xiushan Feng, and Magdy S. Abadir. 2012. Novel Test Detection to Improve Simulation Efficiency - A Commercial Experiment. In *IEEE/ACM International Conference on*

Computer-Aided Design (ICCAD). 101–108.

[25] Ying Chen, Yibo Lin, Tianyang Gai, Yajuan Su, Yayi Wei, and David Z. Pan. 2020. Semisupervised Hotspot Detection With Self-Paced Multitask Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 7 (2020), 1511–1523.

[26] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38, 9 (2019), 1717–1730.

[27] Wei-Kai Cheng, Yu yin Guo, and Chih-Shuan Wu. 2018. Evaluation of Routability-Driven Macro Placement with Machine-Learning Technique. In *International Symposium on Next Generation Electronics*. 1–3.

[28] Vidya A. Chhabria, Andrew B. Kahng, Minsoo Kim, Uday Mallappa, Sachin S. Sapatnekar, and Bangqi Xu. 2020. Template-based PDN Synthesis in Floorplan and Placement Using Classifier and CNN Techniques. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 44–49.

[29] Jason Cong and Yi Zou. 2009. Parallel Multi-Level Analytical Global Placement on Graphics Processing Units. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 681–688.

[30] Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline F. Y. Young, and Zhiru Zhang. 2018. Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 129–132.

[31] Sourav Das, Janardhan Rao Doppa, Daehyun Kim, Partha Pratim Pande, and Krishnendu Chakrabarty. 2015. Optimizing 3D NoC Design for Energy Efficiency: A Machine Learning Approach. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 705–712.

[32] Duo Ding, Jhih-Rong Gao, Kun Yuan, and David Z. Pan. 2011. AENEID: a Generic Lithography-Friendly Detailed Router Based on post-RET Data Learning and Hotspot Detection. In *ACM/IEEE Design Automation Conference (DAC)*. 795–800.

[33] Kerstin Eder, Peter A. Flach, and Hsiou-Wen Hsueh. 2006. Towards Automating Simulation-Based Design Verification Using ILP. In *International Conference on Inductive Logic Programming (Lecture Notes in Computer Science, Vol. 4455)*. 154–168.

[34] Sofiane Ellouz, Patrice Gamand, Christophe Kelma, Bertrand Vandewiele, and Bruno Allard. 2006. Combining Internal Probing with Artificial Neural Networks for Optimal RFIC Testing. In *IEEE International Test Conference (ITC)*. 1–9.

[35] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. 2018. Can Neural Networks Understand Logical Entailment?. In *International Conference on Learning Representations (ICLR)*.

[36] Farnoud Farahmand, Ahmed Ferozpuri, William Diehl, and Kris Gaj. 2017. Minerva: Automated hardware optimization tool. In *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 1–8.

[37] Martin Ferianc, Hongxiang Fan, Ringo S. W. Chu, Jakub Stano, and Wayne Luk. 2020. Improving Performance Estimation for FPGA-Based Accelerators for Convolutional Neural Networks. In *International Symposium on Applied Reconfigurable Computing (ARC) (Lecture Notes in Computer Science, Vol. 12083)*. 3–13.

[38] Shai Fine and Avi Ziv. 2003. Coverage directed test generation for functional verification using bayesian networks. In *ACM/IEEE Design Automation Conference (DAC)*. 286–291.

[39] Evelyn Fix. 1951. *Discriminatory Analysis: Nonparametric Discrimination, Consistency Properties*. USAF School of Aviation Medicine.

[40] Alex Flint and Matthew B. Blaschko. 2012. Perceptron Learning of SAT. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 2780–2788.

[41] Martin Fränzle, Holger Hermanns, and Tino Teige. 2008. Stochastic Satisfiability Modulo Theory: A Novel Technique for the Analysis of Probabilistic Hybrid Systems. In *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC (Lecture Notes in Computer Science, Vol. 4981)*. 172–186.

[42] Alex S. Fukunaga. 2008. Automated Discovery of Local Search Heuristics for Satisfiability Testing. *IEEE Transactions on Evolutionary Computation* 16, 1 (2008), 31–61.

[43] Hao Geng, Haoyu Yang, Yuzhe Ma, Joydeep Mitra, and Bei Yu. 2019. SRAF Insertion via Supervised Dictionary Learning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 406–411.

[44] Hao Geng, Haoyu Yang, Lu Zhang, Jin Miao, Fan Yang, Xuan Zeng, and Bei Yu. 2020. Hotspot Detection via Attention-based Deep Layout Metric Learning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[45] Georges Gielen and Rob Rutenbar. 2001. Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits. *Proc. IEEE* 88 (01 2001), 1825 – 1854.

[46] Cristian Grozea and Marius Popescu. 2014. Can Machine Learning Learn a Decision Oracle for NP Problems? A Test on SAT. *Fundamenta Informaticae* 131, 3-4 (2014), 441–450.

[47] Onur Guzey, Li-C. Wang, Jeremy R. Levitt, and Harry Foster. 2008. Functional Test Selection Based on Unsupervised Support Vector Analysis. In *ACM/IEEE Design Automation Conference (DAC)*. 262–267.

[48] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. 2018. Deep Learning for Logic Optimization Algorithms. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4.

[49] Ali Habibi, Sofiène Tahar, Amer Samarah, Donglin Li, and Otmane Aït Mohamed. 2006. Efficient Assertion Based Verification using TLM. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 106–111.

[50] Shai Haim and Toby Walsh. 2009. Restart Strategy Selection Using Machine Learning Techniques. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5584)*. 312–325.

[51] Kourosh Hakhamaneshi, Nick Werblun, Pieter Abbeel, and Vladimir Stojanovic. 2019. BagNet: Berkeley Analog Generator with Layout Optimizer Boosted with Deep Neural Networks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[52] Seung-Soo Han, Andrew B. Kahng, Siddhartha Nath, and Ashok S. Vydyanathan. 2014. A Deep Learning Methodology to Proliferate Golden Signoff Timing. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 1–6.

[53] Zhuolun He, Yuzhe Ma, Lu Zhang, Peiyu Liao, Ngai Wong, Bei Yu, and Martin DF Wong. 2020. Learn to Floorplan through Acquisition of Effective Local Search Heuristics. In *IEEE International Conference on Computer Design (ICCD)*. IEEE, 324–331.

[54] Chia-Tung Ho and Andrew B. Kahng. 2019. IncPIRD: Fast Learning-Based Prediction of Incremental IR Drop. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[55] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. 1989. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks* 2, 5 (1989), 359–366.

[56] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2020. DRiLLS: Deep Reinforcement Learning for Logic Synthesis. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 581–586.

[57] Kuo-Kai Hsieh, Wen Chen, Li-C. Wang, and Jayanta Bhadra. 2014. On Application of Data Mining in Functional Debug. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 670–675.

[58] Hanbin Hu, Qingran Zheng, Ya Wang, and Peng Li. 2018. HFMV: Hybridizing Formal Methods and Machine Learning for Verification of Analog and Mixed-Signal Circuits. In *ACM/IEEE Design Automation Conference (DAC)*. 95:1–95:6.

[59] Tsung-Wei Huang. 2020. A General-purpose Parallel and Heterogeneous Task Programming System for VLSI CAD. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–2.

[60] Daijoon Hyun, Yuepeng Fan, and Youngsoo Shin. 2019. Accurate Wirelength Prediction for Placement-Aware Synthesis through Machine Learning. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 324–327.

[61] Marketa Illetskova, Alex R. Bertels, Joshua M. Tuggle, Adam Harter, Samuel Richter, Daniel R. Tauritz, Samuel A. Mulder, Denis Bueno, Michelle Leger, and William M. Siever. 2017. Improving Performance of CDCL SAT Solvers by Automated Design of Variable Selection Heuristics. In *IEEE Symposium Series on Computational Intelligence*. 1–8.

[62] Charalambos Ioannides and Kerstin I. Eder. 2012. Coverage-Directed Test Generation Automated by Machine Learning – A Review. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 17, 1, Article 7 (Jan. 2012), 21 pages.

[63] Kwangok Jeong, Andrew B. Kahng, Binshan Lin, and Kambiz Samadi. 2010. Accurate Machine-Learning-Based On-Chip Router Modeling. *IEEE Embedded Systems Letters (ESL)* 2, 3 (2010), 62–66.

[64] Bentian Jiang, Lixin Liu, Yuzhe Ma, Hang Zhang, Bei Yu, and Evangeline F. Y. Young. 2020. Neural-ILT: Migrating ILT to Neural Networks for Mask Printability and Complexity Co-optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–9.

[65] Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. 2020. Efficient Layout Hotspot Detection via Binarized Residual Neural Network Ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[66] Andrew B. Kahng. 2018. New Directions for Learning-based IC Design Tools and Methodologies. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 405–410.

[67] Andrew B. Kahng, Seokhyeong Kang, Hyein Lee, Siddhartha Nath, and Jyoti Wadhwani. 2013. Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools. In *ACM Workshop on System Level Interconnect Prediction (SLIP)*. 1–8.

[68] Andrew B. Kahng, Mulong Luo, and Siddhartha Nath. 2015. SI for free: Machine Learning of Interconnect Coupling Delay and Transition Effects. In *ACM Workshop on System Level Interconnect Prediction (SLIP)*. 1–8.

[69] Yoav Katz, Michal Rimon, Avi Ziv, and Gai Shaked. 2011. Learning Microarchitectural Behaviors to Improve Stimuli Generation Quality. In *ACM/IEEE Design Automation Conference (DAC)*. 848–853.

[70] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 6348–6358.

[71] Ashiqur R. KhudaBukhsh, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2009. SATenstein: Automatically Building Local Search SAT Solvers from Components. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

517–524.

[72] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanovic. 2019. Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1050–1062.

[73] Myung-Chul Kim, Dongjin Lee, and Igor L. Markov. 2010. SimPL: An Effective Placement Algorithm. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 649–656.

[74] Ryan Gary Kim, Janardhan Rao Doppa, and Partha Pratim Pande. 2018. Machine Learning for Design Space Exploration and Optimization of Manycore Systems. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 48.

[75] Bon Woong Ku, Kyungwook Chang, and Sung Kyu Lim. 2018. Compact-2D: A Physical Design Methodology to Build Commercial-Quality Face-to-Face-Bonded 3D ICs. In *ACM International Symposium on Physical Design (ISPD)*. 90–97.

[76] Kishor Kunal, Tonmoy Dhar, Meghna Madhusudan, Jitesh Poojary, Arvind K. Sharma, Wenbin Xu, Steven M. Burns, Jiang Hu, Ramesh Harjani, and Sachin S. Sapatnekar. 2020. GANA: Graph Convolutional Network Based Automated Netlist Annotation for Analog Circuits. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 55–60.

[77] K. Kunal, J. Poojary, T. Dhar, M. Madhusudan, R. Harjani, and S. S. Sapatnekar. [n.d.]. A General Approach for Identifying Hierarchical Symmetry Constraints for Analog Circuit Layout. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[78] Jihye Kwon, Matthew M. Ziegler, and Luca P. Carloni. 2019. A Learning-Based Recommender System for Autotuning Design Flows of Industrial High-Performance Processors. In *ACM/IEEE Design Automation Conference (DAC)*. 218.

[79] Michail G. Lagoudakis and Michael L. Littman. 2001. Learning to Select Branching Rules in the DPLL Procedure for Satisfiability. *Electron. Notes Discret. Math.* 9 (2001), 344–359.

[80] Gil Lederman, Markus N. Rabe, and Sanjit A. Seshia. 2018. Learning Heuristics for Automated Reasoning through Deep Reinforcement Learning. *CoRR* abs/1807.08058 (2018). arXiv:1807.08058

[81] Dongwook Lee and Andreas Gerstlauer. 2018. Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23, 3 (2018), 30:1–30:25.

[82] Bowen Li and Paul D. Franzon. 2016. Machine Learning in Physical Design. In *IEEE Conference on Electrical Performance Of Electronic Packaging And Systems (EPEPS)*. 147–150.

[83] Hao Li, Fanshu Jiao, and Alex Doboli. 2016. Analog Circuit Topological Feature Extraction with Unsupervised Learning of New Sub-Structures. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 1509–1512.

[84] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind K. Sharma, Wenbin Xu, Sachin S. Sapatnekar, Ramesh Harjani, and Jiang Hu. 2020. A Customized Graph Neural Network Model for Guiding Analog IC Placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–9.

[85] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind K. Sharma, Wenbin Xu, Sachin S. Sapatnekar, Ramesh Harjani, and Jiang Hu. 2020. Exploring a Machine Learning Approach to Performance Driven Analog IC Placement. In *IEEE Annual Symposium on VLSI (ISVLSI)*. 24–29.

[86] Jia Liang, Hari Govind V. K., Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. 2018. An Empirical Study of Branching Heuristics through the Lens of Global Learning Rate. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 5319–5323.

[87] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. 2016. Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In *AAAI Conference on Artificial Intelligence*. 3434–3440.

[88] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. 2016. Learning Rate Based Branching Heuristic for SAT Solvers. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9710)*. 123–140.

[89] Rongjian Liang, Hua Xiang, Diwesh Pandey, Lakshmi N. Reddy, Shyam Ramji, Gi-Joon Nam, and Jiang Hu. 2020. DRC Hotspot Prediction at Sub-10nm Process Nodes Using Customized Convolutional Network. In *ACM International Symposium on Physical Design (ISPD)*. 135–142.

[90] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, and G. J. Nam. 2020. Routing-Free Crosstalk Prediction. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[91] Andy Liaw, Matthew Wiener, et al. 2002. Classification and Regression by RandomForest. *R news* 2, 3 (2002), 18–22.

[92] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous Control with Deep Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

[93] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *ACM/IEEE Design Automation Conference (DAC)*. 117.

[94] Dong Liu and Benjamin Carrión Schäfer. 2016. Efficient and Reliable High-Level Synthesis Design Space Explorer for FPGAs. In *IEEE International Conference on Field Programmable Logic and Applications (FPL)*. 1–8.

[95] Hung-Yi Liu and Luca P. Carloni. 2013. On Learning-based Methods for Design-Space Exploration with High-Level Synthesis. In *ACM/IEEE Design Automation Conference (DAC)*. 50:1–50:7.

[96] Mingjie Liu, Wuxi Li, Keren Zhu, Biying Xu, Yibo Lin, Linxiao Shen, Xiyuan Tang, Nan Sun, and David Z. Pan. [n.d.]. S3DET: Detecting System Symmetry Constraints for Analog Circuits with Graph Similarity. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*.

[97] Mingjie Liu, Keren Zhu, Jiaqi Gu, Linxiao Shen, Xiyuan Tang, Nan Sun, and David Z. Pan. 2020. Towards Decrypting the Art of Analog Layout: Placement Quality Prediction via Transfer Learning. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 496–501.

[98] Mingjie Liu, Keren Zhu, Xiyuan Tang, Biying Xu, Wei Shi, Nan Sun, and David Z. Pan. 2020. Closing the Design Loop: Bayesian Optimization Assisted Hierarchical Analog Layout Synthesis. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[99] Siting Liu, Qi Sun, Peiyu Liao, Yibo Lin, and Bei Yu. 2021. Global Placement with Deep Learning-Enabled Explicit Routability Optimization. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*.

[100] Zeye Liu, Qicheng Huang, Chenlei Fang, and R. D. (Shawn) Blanton. 2019. Improving Test Chip Design Efficiency via Machine Learning. In *IEEE International Test Conference (ITC)*. 1–10.

[101] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 20, 2 (2015), 17:1–17:34.

[102] Yi-Chen Lu, Jeehyun Lee, Anthony Agnesina, Kambiz Samadi, and Sung Kyu Lim. 2019. GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[103] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. 2020. TP-GNN: A Graph Neural Network Framework for Tier Partitioning in Monolithic 3D ICs. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[104] Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Lijuan Luo, Karthikeyan Natarajan, and Bei Yu. 2019. High Performance Graph Convolutional Networks with Applications in Testability Analysis. In *ACM/IEEE Design Automation Conference (DAC)*. 18.

[105] Yuzhe Ma, Subhendu Roy, Jin Miao, Jiamin Chen, and Bei Yu. 2018. Cross-layer Optimization for High Speed Adders: A Pareto Driven Machine Learning Approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38, 12 (2018), 2298–2311.

[106] Dani Maarouf, Abeer Alhyari, Ziad Abuowaimer, Timothy Martin, Andrew Gunter, Gary Grewal, Shawki Areibi, and Anthony Vannelli. 2018. Machine-Learning Based Congestion Estimation for Modern FPGAs. In *IEEE International Conference on Field Programmable Logic and Applications (FPL)*. 427–434.

[107] Anushree Mahapatra and Benjamin Carrion Schafer. 2014. Machine-learning based simulated annealer method for high level synthesis design space exploration. *Proceedings of the electronic system level synthesis conference* (2014), 1–6.

[108] Hosein Mohammadi Makrani, Farnoud Farahmand, Hossein Sayadi, Sara Bondi, Sai Manoj Pudukotai Dinakarrao, Houman Homayoun, and Setareh Rafatirad. 2019. Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design. In *IEEE International Conference on Field Programmable Logic and Applications (FPL)*. 397–403.

[109] Hosein Mohammadi Makrani, Hossein Sayadi, Tinoosh Mohsenin, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. 2019. XPPE: Cross-Platform Performance Estimation of Hardware Accelerators Using Machine Learning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 727–732.

[110] Biruk Mammo, Milind Furia, Valeria Bertacco, Scott A. Mahlke, and Daya Shanker Khudia. 2016. BugMD: Automatic Mismatch Diagnosis for Bug Triaging. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 117.

[111] Teruki Matsuba, Nobukazu Takai, Masafumi Fukuda, and Yusuke Kubo. 2018. Inference of Suitable for Required Specification Analog Circuit Topology using Deep Learning. In *2018 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Ishigaki, Okinawa, Japan, November 27-30, 2018*. 131–134.

[112] Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. 2016. Adaptive Threshold Non-Pareto Elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 918–923.

[113] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard C. Ho, Roger Carpenter, and Jeff Dean. 2020. Chip Placement with Deep Reinforcement Learning. *CoRR* abs/2004.10746 (2020). arXiv:2004.10746

[114] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an Efficient SAT Solver. In *ACM/IEEE Design Automation Conference (DAC)*. 530–535.

[115] Walter Lau Neto, Max Austin, Scott Temple, Luca G. Amarù, Xifan Tang, and Pierre-Emmanuel Gaillardon. 2019. LSOracle: a Logic Synthesis Framework Driven by Artificial Intelligence: Invited Paper. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–6.

[116] Kenneth O'Neal, Mitch Liu, Hans Tang, Amin Kalantar, Kennen DeRenard, and Philip Brisk. 2018. HLSPredict: Cross Platform Performance Prediction for FPGA High-level Synthesis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 104.

[117] Jorge Chávez Orzáez, Antonio Jesús Torralba Silgado, and Leopoldo García Franquelo. 1994. A Fuzzy-logic based Tool for Topology Selection in Analog Synthesis. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. 367–370.

[118] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. 2017. Recurrent Relational Networks for Complex Relational Reasoning. *CoRR* abs/1711.08028 (2017). arXiv:1711.08028

[119] Po-Cheng Pan, Chien-Chia Huang, and Hung-Ming Chen. 2019. An Efficient Learning-based Approach for Performance Exploration on Analog and RF Circuit Synthesis. In *ACM/IEEE Design Automation Conference (DAC)*. 232.

[120] Zhijian Pan, Miao Li, Jian Yao, Hong Lu, Zuochang Ye, Yanfeng Li, and Yan Wang. 2018. Low-Cost High-Accuracy Variation Characterization for Nanoscale IC Technologies via Novel Learning-Based Techniques. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 797–802.

[121] Shreepad Panth, Kambiz Samadi, Yang Du, and Sung Kyu Lim. 2017. Shrunk-2-D: A Physical Design Methodology to Build Commercial-Quality Monolithic 3-D ICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36, 10 (2017), 1716–1724.

[122] Sung Joo Park, Bumhee Bae, Joungho Kim, and Madhavan Swaminathan. 2017. Application of Machine Learning for Optimization of 3-D Integrated Circuits and Systems. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 25, 6 (2017), 1856–1865.

[123] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 8024–8035.

[124] Chak-Wa Pui, Gengjie Chen, Yuzhe Ma, Evangeline FY Young, and Bei Yu. 2017. Clock-aware Ultrascale FPGA Placement with Machine Learning Routability Prediction. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 929–936.

[125] Zhongdong Qi, Yici Cai, and Qiang Zhou. 2014. Accurate Prediction of Detailed Routing Congestion Using Supervised Data Learning. In *IEEE International Conference on Computer Design (ICCD)*. 97–103.

[126] Behzad Razavi. 2001. Design of analog CMOS integrated circuits. (01 2001).

[127] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David M. Brooks. 2014. MachSuite: Benchmarks for Accelerator Design and Customized Architectures. In *2014 IEEE International Symposium on Workload Characterization, IISWC 2014, Raleigh, NC, USA, October 26-28, 2014*. 110–119.

[128] Haoxing Ren, George F. Kokai, Walker J. Turner, and Ting-Sheng Ku. 2020. ParaGraph: Layout Parasitics and Device Parameter Prediction using Graph Neural Networks. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[129] João P. S. Rosa, Daniel J. D. Guerra, Nuno C. G. Horta, Ricardo M. F. Martins, and Nuno Lourenço. 2020. *Using ANNs to Size Analog Integrated Circuits*. Springer, 45–66.

[130] Michael Rotman and Lior Wolf. 2020. Electric Analog Circuit Design with Hypernetworks And A Differential Simulator. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4157–4161.

[131] Sandeep Kumar Samal, Guoqing Chen, and Sung Kyu Lim. 2016. Machine Learning Based Variation Modeling and Optimization for 3D ICs. *Journal of Information and Communication Convergence Engineering* 14, 4 (2016).

[132] Daniel Selsam and Nikolaj Bjørner. 2019. NeuroCore: Guiding High-Performance SAT Solvers with Unsat-Core Predictions. *CoRR* abs/1903.04671 (2019). arXiv:1903.04671

[133] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT Solver from Single-Bit Supervision. In *International Conference on Learning Representations (ICLR)*.

[134] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. 2020. AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 490–495.

[135] Haihua Shen, Wenli Wei, Yunji Chen, Bowen Chen, and Qi Guo. 2008. Coverage Directed Test Generation: Godson Experience. In *IEEE Asian Test Symposium (ATS)*. 321–326.

[136] Brett Shook, Prateek Bhansali, Chandramouli Kashyap, Chirayu Amin, and Siddhartha Joshi. 2020. MLParest: Machine Learning based Parasitic Estimation for Custom Circuit Design. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[137] Antonio Jesús Torralba Silgado, Jorge Chávez Orzáez, and Leopoldo García Franquelo. 1996. FASY: a Fuzzy-Logic Based Tool for Analog Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 15, 7 (1996), 705–715.

[138] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the Game of Go without Human Knowledge. *Nat.* 550, 7676 (2017), 354–359.

[139] Rishabh Singh, Joseph P Near, Vijay Ganesh, and Martin Rinard. 2009. Avatarsat: An Auto-Tuning Boolean Sat Solver. (2009).

[140] Haralampos-G. D. Stratigopoulos, Petros Drineas, Mustapha Slamani, and Yiorgos Makris. 2007. Non-RF to RF Test Correlation Using Learning Machines: A Case Study. In *IEEE VLSI Test Symposium (VTS)*. 9–14.

[141] Haralampos-G. D. Stratigopoulos, Petros Drineas, Mustapha Slamani, and Yiorgos Makris. 2010. RF Specification Test Compaction Using Learning Machines. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 18, 6 (2010), 998–1002.

[142] Haralampos-G. D. Stratigopoulos and Yiorgos Makris. 2008. Error Moderation in Low-Cost Machine-Learning-Based Analog/RF Testing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 27, 2 (2008), 339–351.

[143] Ecenur Ustun, Chenhui Deng, Debjit Pal, Zhijing Li, and Zhiru Zhang. 2020. Accurate Operation Delay Prediction for FPGA HLS Using Graph Neural Networks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–9.

[144] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 2692–2700.

[145] Ilya Wagner, Valeria Bertacco, and Todd M. Austin. 2007. Microprocessor Verification via Feedback-Adjusted Markov Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 26, 6 (2007), 1126–1138.

[146] Fanchao Wang, Hanbin Zhu, Pranjay Popli, Yao Xiao, Paul Bogdan, and Shahin Nazarian. 2018. Accelerating Coverage Directed Test Generation for Functional Verification: A Neural Network-based Framework. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*. 207–212.

[147] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[148] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. 2018. Learning to Design Circuits. *CoRR* abs/1812.02734 (2018). arXiv:1812.02734

[149] Zi Wang and Benjamin Carrión Schäfer. 2020. Machine Leaming to Set Meta-Heuristic Specific Parameters for High-Level Synthesis Design Space Exploration. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[150] Samuel I. Ward, Duo Ding, and David Z. Pan. 2012. PADE: a High-Performance Placer with Automatic Datapath Extraction and Evaluation Through High Dimensional Data Learning. In *ACM/IEEE Design Automation Conference (DAC)*. 756–761.

[151] Samuel I. Ward, Myung-Chul Kim, Natarajan Viswanathan, Zhuo Li, Charles J. Alpert, Earl E. Swartzlander Jr., and David Z. Pan. 2012. Keep it Straight: Teaching Placement How to Better Handle Designs with Datapaths. In *ACM International Symposium on Physical Design (ISPD)*. 79–86.

[152] Po-Hsun Wu, Mark Po-Hung Lin, and Tsung-Yi Ho. 2015. Analog Layout Synthesis with Knowledge Mining. In *European Conference on Circuit Theory and Design (ECCTD)*. 1–4.

[153] Zhiyao Xie, Guan-Qi Fang, Yu-Hung Huang, Haoxing Ren, Yanqing Zhang, Brucek Khailany, Shao-Yun Fang, Jiang Hu, Yiran Chen, and Erick Carvajal Barboza. 2020. FIST: A Feature-Importance Sampling and Tree-Based Method for Automatic Design Flow Parameter Tuning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 19–25.

[154] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability Prediction for Mixed-Size Designs Using Convolutional Neural Network. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 80.

[155] Zhiyao Xie, Haoxing Ren, Brucek Khailany, Ye Sheng, Santosh Santosh, Jiang Hu, and Yiran Chen. 2020. PowerNet: Transferable Dynamic IR Drop Estimation via Maximum Convolutional Neural Network. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 13–18.

[156] Biying Xu, Yibo Lin, Xiyuan Tang, Shaolan Li, Linxiao Shen, Nan Sun, and David Z. Pan. 2019. WellGAN: Generative-Adversarial-Network-Guided Well Generation for Analog/Mixed-Signal Circuit Layout. In *ACM/IEEE Design Automation Conference (DAC)*. 66.

[157] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. SATzilla: Portfolio-based Algorithm Selection for SAT. *CoRR* abs/1111.2249 (2011). arXiv:1111.2249

[158] Xiaoqing Xu, Tetsuaki Matsunawa, Shigeki Nojima, Chikaaki Kodama, Toshiya Kotani, and David Z. Pan. 2016. A Machine Learning Based Framework for Sub-Resolution Assist Feature Generation. In *ACM International Symposium on Physical Design (ISPD)*. 161–168.

[159] Haoyu Yang, Shuhe Li, Yuzhe Ma, Bei Yu, and Evangeline F. Y. Young. 2018. GAN-OPC: Mask Optimization with Lithography-Guided Generative Adversarial Nets. In *ACM/IEEE Design Automation Conference (DAC)*. 131:1–131:6.

[160] Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, and Bei Yu. 2017. Imbalance Aware Lithography Hotspot Detection: a Deep Learning Approach. *Journal of Micro/Nanolithography, MEMS, and MOEMS* 16, 3 (2017), 033504.

[161] Haoyu Yang, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu. 2019. Detecting Multi-layer Layout Hotspots with Adaptive Squish Patterns. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 299–304.

[162] Haoyu Yang, Jing Su, Yi Zou, Bei Yu, and Evangeline F. Y. Young. 2017. Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning. In *ACM/IEEE Design Automation Conference (DAC)*. 62:1–62:6.

[163] Haoyu Yang, Wei Zhong, Yuzhe Ma, Hao Geng, Ran Chen, Wanli Chen, and Bei Yu. 2020. VLSI Mask Optimization: From Shallow To Deep Learning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 434–439.

[164] Que Yanghua, Nachiket Kapre, Harnhua Ng, and Kirvy Teo. 2016. Improving Classification Accuracy of a Machine Learning Approach for FPGA Timing Closure. In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 80–83.

[165] Wei Ye, Mohamed Baker Alawieh, Yibo Lin, and David Z. Pan. 2019. LithoGAN: End-to-End Lithography Modeling with Generative Adversarial Networks. In *ACM/IEEE Design Automation Conference (DAC)*. 107.

[166] Emre Yolcu and Barnabás Póczos. 2019. Learning Local Search Heuristics for Boolean Satisfiability. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 7990–8001.

[167] Cunxi Yu, Houping Xiao, and Giovanni De Micheli. 2018. Developing Synthesis Flows without Human Knowledge. In *ACM/IEEE Design Automation Conference (DAC)*. 50:1–50:6.

[168] Guo Zhang, Hao He, and Dina Katabi. 2019. Circuit-GNN: Graph Neural Networks for Distributed Circuit Design. In *International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research, Vol. 97)*. 7364–7373.

[169] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. GRANNITE: Graph Neural Network Inference for Transferable Power Estimation. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[170] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. Opportunities for RTL and Gate Level Simulation using GPUs (Invited Talk). In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–5.

[171] Chen Zhao, Zhenya Zhou, and Dake Wu. 2020. Empyrean ALPS-GT: GPU-Accelerated Analog Circuit Simulation. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–3.

[172] Jieru Zhao, Tingyuan Liang, Sharad Sinha, and Wei Zhang. 2019. Machine Learning Based Routing Congestion Prediction in FPGA High-Level Synthesis. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 1130–1135.

[173] Han Zhou, Wentian Jin, and Sheldon X.-D. Tan. 2020. GridNet: Fast Data-Driven EM-Induced IR Drop Prediction and Localized Fixing for On-Chip Power Grid Networks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–9.

[174] Yuan Zhou, Udit Gupta, Steve Dai, Ritchie Zhao, Nitish Kumar Srivastava, Hanchen Jin, Joseph Featherston, Yi-Hsiang Lai, Gai Liu, Gustavo Angarita Velasquez, Wenping Wang, and Zhiru Zhang. 2018. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs. In *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*. 269–278.

[175] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. PRIMAL: Power Inference using Machine Learning. In *ACM/IEEE Design Automation Conference (DAC)*. 39.

[176] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. 2018. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) (Lecture Notes in Computer Science, Vol. 11045)*. 3–11.

[177] Keren Zhu, Mingjie Liu, Yibo Lin, Biying Xu, Shaolan Li, Xiyuan Tang, Nan Sun, and David Z. Pan. 2019. GeniusRoute: A New Analog Routing Paradigm Using Generative Neural Network Guidance. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[178] Cheng Zhuo, Bei Yu, and Di Gao. 2017. Accelerating Chip Design with Machine Learning: From pre-Silicon to post-Silicon. In *IEEE International System-on-Chip Conference (SOCC)*. 227–232.

[179] Matthew M. Ziegler, Hung-Yi Liu, and Luca P. Carloni. 2016. Scalable Auto-Tuning of Synthesis Parameters for Optimizing High-Performance Processors. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 180–185.

[180] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. 2013. Active Learning for Multi-Objective Optimization. In *International Conference on Machine Learning (ICML) (JMLR Workshop and Conference Proceedings, Vol. 28)*. 462–470.