SCIENCE CHINA
Information Sciences

• REVIEW •

November 2023, Vol. 66 211101:1–211101:31
https://doi.org/10.1007/s11432-022-3772-8

Chip design with machine learning: a survey from algorithm perspective

Wenkai HE[1,2,3], Xiaqing LI[1], Xinkai SONG[1,3], Yifan HAO[1,3], Rui ZHANG[1,3], Zidong DU[1] & Yunji CHEN[1,2*]

[1]*State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190*, China;*
[2]*University of Chinese Academy of Sciences, Beijing* 100049*, China;*
[3]*Cambricon Technologies, Beijing* 100191*, China*

Received 7 December 2022/Revised 15 March 2023/Accepted 16 May 2023/Published online 19 October 2023

Abstract Chip design with machine learning (ML) has been widely explored to achieve better designs, lower runtime costs, and no human-in-the-loop process. However, with tons of work, there is a lack of clear links between the ML algorithms and the target problems, causing a huge gap in understanding the potential and possibility of ML in future chip design. This paper comprehensively surveys existing studies in chip design with ML from an algorithm perspective. To achieve this goal, we first propose a novel and systematical taxonomy that divides target problems in chip design into three categories. Then, to solve the target problems with ML algorithms, we formulate the three categories as three ML problems correspondingly. Based on the taxonomy, we conduct a comprehensive survey in terms of target problems based on different ML algorithms. Finally, we conclude three key challenges for existing studies and highlight several insights for the future development of chip design with machine learning. By constructing a clear link between chip design problems and ML solutions, we hope the survey can shed light on the road to chip design intelligence from previous chip design automation.

Keywords chip design, machine learning, chip design automation, design result estimation, design optimization and correction, design construction

Citation He W K, Li X Q, Song X K, et al. Chip design with machine learning: a survey from algorithm perspective. Sci China Inf Sci, 2023, 66(11): 211101, https://doi.org/10.1007/s11432-022-3772-8

1 Introduction

Over the past several years, machine learning (ML) algorithms have been widely utilized in chip design for their potential to enhance design quality, reduce runtime costs, and achieve no human-in-the-loop process [1–7]. ML algorithms, especially recent deep learning (DL) algorithms [8–11] and reinforcement learning (RL) algorithms [12,13], have been extensively explored to address various problems encountered in chip design, such as high-level synthesis (HLS) [14–24], floorplanning [25–37], placement [38–61], and routing [62–69]. Despite the abundance of research in this area, there remains a lack of clear links between the ML algorithms and the target problems in chip design, causing a huge gap in understanding the potential and possibility of ML in future chip design.

This paper comprehensively surveys existing studies of chip design with ML algorithms from an algorithm perspective. To accomplish this goal, we propose a novel and systematical taxonomy for the target problems at different stages in chip design. The taxonomy aims to guide selecting and designing ML algorithms for the target problems, taking into account their existing challenges, as well as to provide a comprehensive summary of chip design with ML algorithms for researchers. As illustrated in Figure 1, we classify the target problems in chip design into three categories: design result estimation, design optimization and correction, and design construction, which are commonly encountered in different stages of logic design, circuit design, and physical design, as well as in verification and test of each stage. Specifically, design result estimation encompasses problems that predict or estimate design quality,

* Corresponding author (email: cyj@ict.ac.cn)

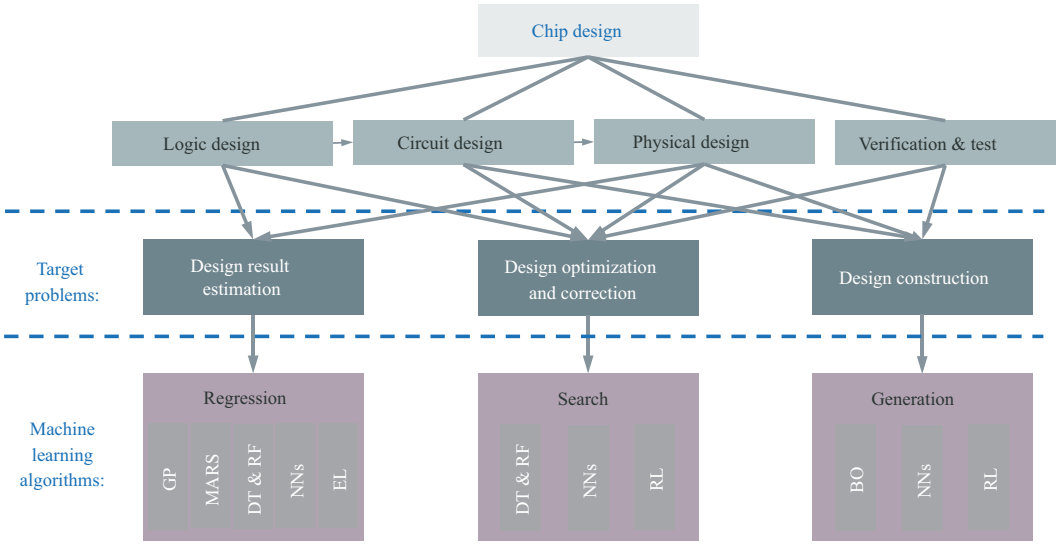

Figure 1 (Color online) Overall taxonomy.

such as performance estimation in logic design, IR drop estimation in physical design, and static timing analysis (STA) in logic design and physical design. Design optimization and correction refer to problems that optimize design quality and correct design errors, respectively, such as HLS design space exploration (DSE) in logic design, logic optimization in circuit design, and detailed routing in physical design. Design construction encompasses problems that generate design representations or add the design objects (i.e., components and routes), such as physical mapping in circuit design and global placement in physical design.

To address these target problems with ML algorithms, we formulate the three categories of the target problems as three corresponding problems in ML: regression, search, and generation, as shown in Figure 1. Each of these ML problems can be tackled by various ML algorithms. Specifically, the design result estimation problem is formulated as a regression problem and can be addressed using several ML algorithms, including Gaussian process (GP), multivariate adaptive regression splines (MARS), decision tree (DT) & random forest (RF), neural networks (NNs), and ensemble learning (EL). The design optimization and correction problem is formulated as a search problem and can be tackled using various ML algorithms, including DT & RF, NNs, and RL. Finally, the design construction problem is formulated as a generation problem and can be tackled using various ML algorithms, including Bayesian optimization (BO), NNs, and RL.

Based on the taxonomy, we comprehensively survey the existing studies in terms of the target problems by the following steps. Our approach is structured as follows: firstly, we present a definition for each target problem and analyze the inherent reasons for applying ML algorithms in addressing these target problems. Secondly, we conduct a comprehensive survey in terms of target problems according to the ML algorithms. We introduce specific ML-based work for each target problem and explain how they improve upon the origin chip design tools based on conventional algorithms. Finally, we conclude by highlighting three key challenges that remain unresolved in the existing studies and provide several insights for future research in chip design with ML algorithms, including single-stage end-to-end generation, cross-stage end-to-end generation, whole-process end-to-end generation, practicability improvement, and others, which we hope to boost the research in chip design with ML. By establishing a clear link between chip design problems and corresponding ML solutions, our survey aims to shed light on the road to chip design intelligence from previous chip design automation.

To the best of our knowledge, this paper is the first work to comprehensively survey chip design with ML from an algorithm perspective. We summarize the main contributions of this survey as follows.

(1) Deep analysis of chip design. We dissect the commonly adopted process of chip design and analyze the key steps in different design stages (i.e., logic design, circuit design, and physical design) as well as in verification and test of each stage, where the key design, advantages, and disadvantages of the steps are analyzed.

(2) Innovative taxonomy. We classify target problems into three categories (i.e., design result

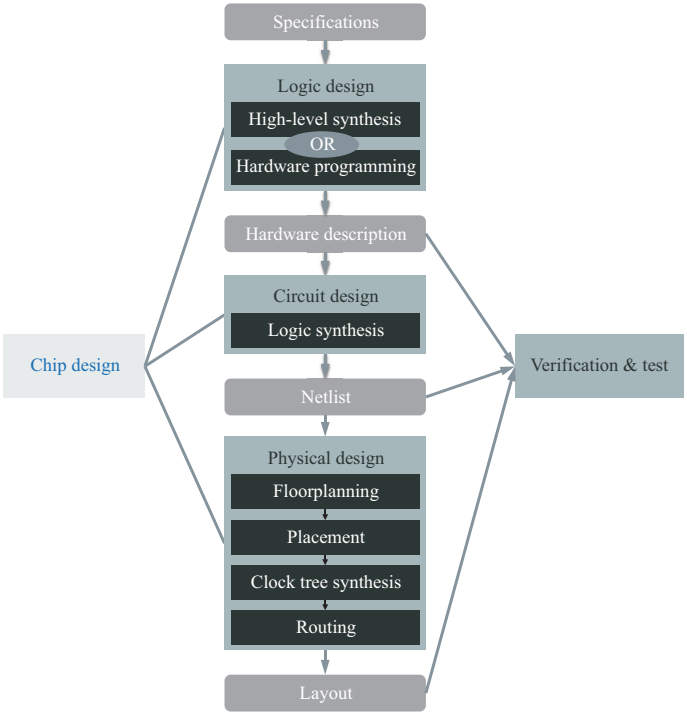

Figure 2 (Color online) Chip design process.

estimation, design optimization and correction, and design construction) and further formulate them into three ML problems, respectively: regression, search, and generation.

(3) Comprehensive survey. Based on the taxonomy, we present the definition of each target problem and analyze the inherent reasons why ML algorithms are used to address the target problems. We also conduct a comprehensive survey in terms of target problems based on different ML algorithms.

(4) Future work. Finally, we conclude by highlighting three key challenges that remain unsolved in existing studies and provide insights into the future development of chip design with ML algorithms, which we hope to boost the research in chip design with ML.

The rest of this paper is organized as follows. In Section 2, we analyze the chip design process, provide an overview of ML, and present the taxonomy from an algorithmic perspective. Sections 3–5 provide a comprehensive survey of existing studies in chip design with ML algorithms, covering design result estimation with regression ML algorithms (Section 3), design optimization and correction with search ML algorithms (Section 4), and design construction with generation ML algorithms (Section 5). In Section 6, we highlight three key challenges that remain unsolved in chip design with ML, and we suggest several future work to tackle these challenges.

2 Chip design and machine learning

2.1 A primer on chip design

After four decades of development, chip design has witnessed the integration of various software tools that incorporate advanced algorithms to facilitate design automation, commonly referred to as electronic design automation (EDA) tools. As depicted in Figure 2, the chip design can be roughly divided into three stages: logic design, circuit design, and physical design. Besides, verification and test are carried out at every stage to guarantee correctness. It is worth noting that in this survey, we focus exclusively on digital circuit design.

2.1.1 *Logic design*

The primary objective of the logic design stage is to generate the logic description for a chip starting from scratch. Typically, this is achieved by either automatically generating or manually developing register-

transfer-level (RTL) hardware codes such as Verilog or VHDL. Automatic generation is commonly referred to as HLS, while manual development is known as hardware programming.

HLS automatically transforms hardware functions to RTL programs through HLS tools, adhering to user-specified design constraints. Typically, hardware functions are described by high-level programming languages such as C, C++, and SystemC, and RTL programs are characterized as hardware codes such as Verilog or VHDL. Generally, traditional HLS algorithms utilize deterministic algorithms to perform these transformations [1]. However, on a large-scale design, such transformations can be excessively time-consuming due to the very large and highly complex space, where the design complexity grows exponentially with the length of the RTL program.

Hardware programming manually writes the RTL programs, which are characterized as hardware description languages (HDL), such as Verilog and VHDL. HDLs describe the behavior and structure of electronic circuits, particularly digital logic circuits. Verilog, for example, models electronic circuits, particularly digital circuit design and verification, at the RTL of abstraction. Although hardware programming typically requires more time than HLS, it remains popular among chip designers because it leverages human experience and can achieve better design quality than HLS tools.

2.1.2 *Circuit design*

After the hardware description (i.e., an RTL program) is determined, the circuit design is conducted to obtain the circuit-level, typically gate-level, description of the design, which is also known as logic synthesis or front-end design in chip design.

Logic synthesis is a process in which the chip designer transforms the RTL program into a netlist, a gate-level circuit representation that is characterized as an HDL. This process primarily comprises two core steps: logic optimization and physical mapping. The former step aims to simplify the circuit's Boolean expression and logical netlist structure to obtain a logical expression with the simplest possible logic. The latter step aims to generate the simplest possible physical circuit by mapping the logical expression to physical units for a given technology library, which defines the types of available gates and standard cells. Effective utilization of logic synthesis enables the construction of a netlist without any physical information, which is completed in subsequent processes. However, in practice, both steps are challenging to identify the optimal logic expression and netlist due to the highly complex search space in logic optimization [1, 70] and the poor generalization of determining the optimal solution in physical mapping [71].

2.1.3 *Physical design*

Once the netlist is established, the physical design is conducted to obtain the placed and routed layout of the design, which is suitable for delivery to the foundry for the purpose of tape-out. The physical design typically contains multiple stages such as floorplanning, placement, clock tree synthesis (CTS), and routing.

Floorplanning determines the initial physical information (i.e., physical design environment and parameters) for a layout and places instantiated modules and macros on the layout while considering physical constraints. A well-executed floorplanning can generate a general region plan for placing standard cells and is a critical requirement for successful placement deployment. Unfortunately, achieving desirable quality floorplanning often requires expert knowledge and collaboration across various fields, leading to high development costs and prolonged production cycles.

Placement assigns logic components (i.e., standard cells) on the layout based on the given netlist from logic synthesis and available standard cell types in a technology library. Accurate placement is crucial for improving design performance and routability, ultimately impacting chip manufacturability. However, existing placement approaches still face three challenges. First, the optimal placed layout cannot be theoretically obtained because the quality of placement is heavily dependent on the posterior unknown routing strategy. The main reason is that the complexity of global placement, gate sizing, and power delivery network (PDN) synthesis makes cross-step or global optimization difficult. Second, placement is time-consuming, as global placement involves placing many standard cells with a complexity greater than exponential, leading to a long runtime. Finally, gate sizing in placement requires careful consideration of netlist context details [60] and often relies on expert knowledge, leading to a high dependency on human participation.

CTS is a process that aims to balance the delay of clock signals to all clock inputs by inserting buffers or inverters along the clock paths of chip design, with the goal of minimizing the skew and insertion delay. This optimization is crucial because it can significantly improve the maximum achievable clock frequency (i.e., throughput) and overall chip performance. However, analyzing the skew can be a time-intensive process, requiring the analysis of many paths and all possible design cases. Therefore, CTS is often time-consuming, especially for large-scale design tasks.

Routing, or wire routing, is a crucial process that allocates routing resources for wires and establishes connections between multiple components (i.e., macros and standard cells) based on specific design rules. This process is critical for chip design as it ensures functional correctness and can significantly impact chip performance and signal integrity (SI). However, existing routing approaches face two problems. Firstly, global routing is time-consuming due to the exponential design space complexity relative to the number of possible wires. Secondly, detailed routing requires human participation to check and eliminate congestion with high accuracy.

2.1.4 *Verification and test*

Verification and test run through the whole process of chip design and are performed iteratively to ensure the correctness of each stage's output. This helps to save significant costs for chip designers and avoids expensive trial and error in the design and manufacturing phases. Typical verification and test are complicated and involve different types, including functional verification and test, logic verification and test, circuit verification and test, netlist verification and test, and layout verification and test. Unfortunately, each type of verification and test is time-consuming, and multiple iterations of these steps for a single design task require a substantial amount of time and effort.

2.2 A primer on machine learning

As a branch of artificial intelligence (AI), ML focuses on enabling machines to 'learn' from data like human beings. Typically, ML algorithms construct a model that utilizes sample data to enhance the accuracy of predictions or decisions. In recent years, ML algorithms have demonstrated remarkable success in a wide range of challenging tasks across various industries and academic domains, such as computer vision (CV) [10], natural language processing (NLP) [72], automated decision-making [73], bioinformatics [74], and mathematics [75]. Nowadays, ML is the dominant force in the field of AI and has brought prosperity to AI.

ML has also proven to be successful in all stages of chip design. The common target problems encountered in chip design can be roughly formulated into three core ML problems: regression, search, and generation. Consequently, numerous ML algorithms can be employed to address these three core problems and enhance the performance of chip design.

Regression. Regression aims to learn the parameters of a model to estimate the relationship between independent and dependent variables. The independent variables are commonly input data, such as raw data or hand-crafted features, while dependent variables refer to ground truth, such as labels annotated by humans. Regression models are trained on input-output samples, and given input data, the model generates predictions of the ground truth. Regression algorithms aim to minimize the distance between predictions and ground truth. In chip design, regression algorithms are used to solve design result estimation problems. The independent variables are extracted features of design at an exact step, including chip specifications, design rules, and the current design. The regression model aims to estimate design-related judgment, including quality, timing, performance, and resource costs. The produced design-related judgments are then used to validate the design and guide design optimization. To improve the accuracy and efficiency of regression, various ML algorithms, such as GP [14], MARS [62,63], DT & RF [15,16,25,38,39], NNs [17,26,40–49,76–80], and EL [18–20,27,28,50,51], have been applied.

Search. Search intelligently finds the optimal solution from a search space that includes all possible solutions. Simple exhaustive search can be inefficient and slow because the number of search steps grows quickly in the large search space. To address this issue, heuristic search algorithms have been developed with two goals. First, some heuristic search algorithms prioritize solutions that are more likely to be optimal. Second, some heuristic algorithms eliminate some solutions that are unlikely to lead to optimal solutions. In chip design, search algorithms are used for design optimization and correction problems, where they read the design and iteratively explore the design space to discover a feasible and superior design. The start points of search algorithms are designed by humans or EDA tools. These initial designs

often have limitations or faults, including poor power, performance, and area (PPA), as well as functional faults and design failures. Search algorithms iteratively tune the design to improve the quality of PPA, correct functional faults, and find a design without failure. To accelerate the search process and improve the quality of solutions, ML algorithms, such as DT & RF [21, 22, 29], NNs [23, 52–54, 64, 65, 81–83], and RL [24, 30, 55–57, 66–68, 70, 84–86], are applied to estimate the appropriate search direction.

Generation. Generation focuses on predicting the target variable from the given observation variable. This is accomplished by learning a generative model to build the joint probability distribution of the observable and target variables and then using this model to estimate the target variable given the observation variable. Nowadays, generation algorithms have shown impressive results in various fields such as images [9], texts [72], and programs [87]. In chip design, generation algorithms are applied in two ways. The first is to generate the representation of the design for the next stage based on the current stage. For example, physical mapping generates a gate-level circuit representation from an RTL representation of a circuit. The second is to add objects relevant to the next stage of the current design, such as standard cells added during global placement. To improve the efficiency and quality of the generation process, multiple ML algorithms, including BO [31, 71], NNs [32, 33, 58–61, 69, 88–90], and RL [34–37], have been applied to solve the generation problem.

2.3 Taxonomy: from the algorithm perspective

In this paper, we survey and summarize studies of chip design with ML in taxonomy as depicted in Figure 1. Chip design is divided into logic design, circuit design, physical design, as well as verification and test, and the target problems in them are classified into three types: design result estimation, design optimization and correction, and design construction. Based on our survey, design result estimation problems can be formalized as regression problems, design optimization and correction problems can be formalized as search problems, and design construction problems can be formalized as generation problems.

Our taxonomy starts from an algorithm perspective for two reasons. Firstly, while chip design may consist of various stages and problems, many of these can be reformulated as similar issues from an ML perspective and thus can be addressed using similar ML algorithms. For example, estimating the design quality is the same problem, where in logic design, the input is RTL designs [14–20], and in physical design, the placed and routed layouts [26, 38–51, 62, 63, 78–80]. These problems can be formalized as regression problems and solved with regression ML algorithms, such as the RF [15, 16, 25, 38, 39] and NNs [17, 26, 40–49, 76–80]. Secondly, reviewing and surveying studies based on tasks or functionality have already been extensively explored [1–5]. Currently, for leveraging ML algorithms (especially emerging DL and RL) in chip design, the most significant barrier is how to formalize a given chip design problem as an ML problem. This survey aims to build links between them to help chip designers and chip design researchers understand why a particular ML algorithm can be used to solve chip design problems, which we think could be more helpful.

The rest of the paper introduces design result estimation problems solved with regression ML algorithms, design optimization and correction problems solved with search ML algorithms, and design construction problems solved with generation ML algorithms, respectively. All the studies introduced below are summarized in Table 1.

3 Design result estimation with regression machine learning algorithms

3.1 Design result estimation problems

Design result estimation refers to problems that predict or estimate the quality of design. These problems read the design and judge the current design in different stages for different purposes, as shown in Table 2. In the logic design stage, design result estimation includes performance estimation, STA, quality estimation, and routability estimation. In the physical design stage, design result estimation includes STA, quality estimation, routablity estimation, IR drop estimation, bump inductance prediction, coupling effect prediction, net length prediction, clock power estimation, and SI analysis.

Performance estimation. Performance estimation aims to estimate a chip design's throughput and throughput-to-area ratio, particularly in the logic design stage. Typically, the throughput is estimated by the maximum achievable clock frequency from an HLS timing report. The estimated result is then

Table 1 Taxonomy and corresponding papers[a)]

Stage	Target problem category	Target problem	ML algorithm	Ref.
Logic design	Design result estimation	Performance estimation	GP	[14]
			NNs	[17]
			EL	[18]
		Static timing analysis	NNs	[19]
			EL	[20]
		Quality estimation	DT & RF	[15]
		Routability estimation	DT & RF	[16]
	Design optimization and correction	HLS design space exploration	DT & RF	[21, 22]
			NNs	[23]
			RL	[24]
	Design construction	–	–	–
Circuit design	Design result estimation	–	–	–
	Design optimization and correction	Logic optimization	NNs	[81]
			RL	[70, 84, 85]
	Design construction	Physical mapping	BO	[81]
Physical design	Design result estimation	Static timing analysis	DT & RF	[38]
			NNs	[40, 41]
			EL	[27, 28]
		Routability estimation	MARS	[62, 63]
			DT & RF	[25]
			NNs	[26, 42–47]
		IR drop estimation	DT & RF	[39]
			NNs	[48, 49]
		Bump inductance prediction	EL	[50]
		Coupling effect prediction	NNs	[79]
		Net length prediction	NNs	[80]
			EL	[51]
		Clock power estimation	NNs	[78]
		Signal integrity analysis	NNs	[76, 77]
	Design optimization and correction	Floorplanning optimization	DT & RF	[29]
			RL	[30]
		Gate sizing	NNs	[54]
			RL	[57]
		Detailed routing	RL	[66–68]
		Layout optimization	NNs	[52, 53, 64, 65]
			RL	[55, 56]
	Design construction	Module placement	BO	[31]
			NNs	[32, 33]
			RL	[34–37]
		Global placement	NNs	[58, 59]
		Gate sizing	NNs	[60]
		PDN synthesis	NNs	[61]
		Clock tree synthesis	NNs	[88]
		Global routing	NNs	[69]
Verification & test	Design result estimation	–	–	–
	Design optimization and correction	Test point insertion	NNs	[82]
			RL	[86]
		Test metrics optimization	NNs	[83]
	Design construction	Testbench generation	NNs	[90]

a) "–" shows that this paper does not mention any problem in this category, as is the case with subsequent tables.

used to optimize the design to meet the required performance. Although the performance estimation in logic design is generally less precise than in circuit design and physical design, it is still valuable for facilitating rapid design iteration.

Table 2 Design result estimation problems in chip design

	Design result estimation	Problem description	Input	Output
Logic design	Performance estimation	Estimate the throughput	HLS reports	Throughput
	Static timing analysis	Measure latencies	RTL designs	Latencies
	Quality estimation	Measure hardware characteristics	HLS reports	QoRs
	Routability estimation	Measure routing possibility	RTL designs	Congestion
Circuit design	–	–	–	–
Physical design	Static timing analysis	Measure latencies	Layouts	Latencies
	Routablity estimation	Measure routing possibility	Layouts	Congestion
	IR drop estimation	Evaluate power levels	Layouts	IR drop/EM
	Bump inductance prediction	Estimate PDN quality	Layouts	Bump inductance
	Coupling effect prediction	Estimate coupling capacitance	Layouts	Coupling capacitance
	Net length prediction	Analyze net lengths	Layouts	Net lengths
	Clock power estimation	Predict clock network's power	Layouts	Power
	Signal integrity analysis	Evaluate quality of digital signal	Layouts	EW/EH/BER
Verification & test	–	–	–	–

Static timing analysis. Static timing analysis measures the delay of all paths in the design to ensure there are no timing violations, avoiding the need for costly and time-consuming full simulation. STA is an essential step in every chip design stage to validate the design after transformation or optimization by EDA tools. Even without a full simulation, the cost of STA is still not negligible.

Quality estimation. Quality estimation, or quality of results (QoRs) estimation, measures the hardware characteristics of a design, including resource usage (especially area), timing, and power. It is crucial in every chip design stage to assess whether the final chip meets the given constraints, such as chip size, clock frequency, and power. However, quality estimation in the earlier stages is more challenging and less accurate than in later stages.

Routability estimation. Routability estimation measures the possibility of whether a design can be successfully routed after placement in the physical design stage. By estimating the routing result instead of running computationally-expensive routers, routability estimation can help save time and resources. However, it remains a computationally-intensive problem, and accurate estimation is still very challenging, as routability is dependent on the router used, and each router is highly complex.

IR drop estimation. IR drop estimation evaluates IR drop (i.e., deviation of the power level from its specification) and ElectroMigration (EM) to determine whether a circuit meets its timing and function constraints. Accurate IR drop estimation can reduce the turnaround time among module placement, PDN synthesis, and global placement [39], leading to faster design convergence. However, accurate IR drop estimation methods can be time-consuming, especially as chip scale and complexity continue to grow.

Bump inductance prediction. Bump inductance prediction measures the bump inductance on a layout to estimate the quality of the PDN. The quality of the PDN is critical for ensuring that sufficient power can be delivered to critical blocks in a chip [50], thus guaranteeing its functional correctness. Unfortunately, current bump inductance prediction methods have low accuracy, resulting in a longer turnaround time between PDN synthesis and manual optimization.

Coupling effect prediction. Coupling effect prediction measures the coupling capacitance of a design to estimate crosstalk effects, which is performed prior to routing. Accurate crosstalk effect estimation can improve the robustness of PPA optimization [79]. However, as the router determines coupling capacitance, estimating crosstalk effects before routing lacks adequate information, which requires numerous iterations between placement and routing.

Net length prediction. Net length prediction measures net lengths before routing to estimate routing resource usage and critical path lengths. Accurate net length prediction ensures accurate timing [80] or power [51] estimation and optimization. Unfortunately, as net lengths are determined during the

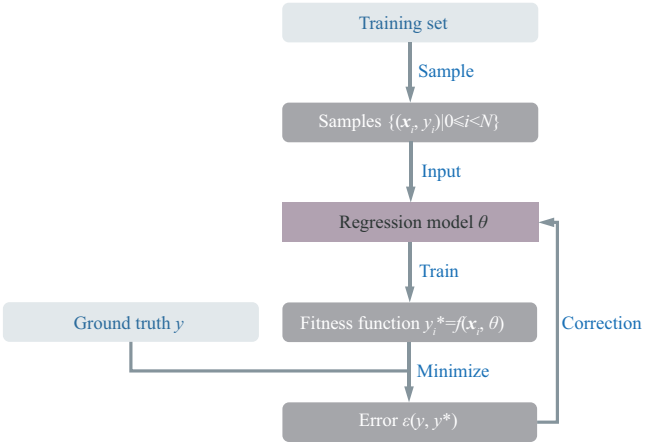

Figure 3 (Color online) Framework of the regression algorithm.

routing stage, predicting net lengths beforehand suffers from inadequate information. Therefore, achieving accurate net length prediction prior to routing poses a significant challenge.

Clock power estimation. Clock power estimation is to predict the power consumption of the clock network before CTS. Clock power estimation is crucial in determining the overall power consumption of a chip as the clock network is often the main power consumer in chip design [91]. However, accurately estimating the power consumption of the clock network before CTS is challenging as it depends on the clock network's structure generated by CTS.

Signal integrity analysis. Signal integrity analysis evaluates the quality of the digital signal by measuring the eye height (EH), eye width (EW), and bit error rate (BER). High-quality digital signals ensure the reliability of a chip. However, SI analysis is computationally expensive and requires significant resources, such as power, memory, and time [76, 92].

Summary. It can be summarized that accuracy and speed are the two most significant concerns in the aforementioned design result estimation problems. Design result estimation problems require fast and accurate estimation to enable iterative design. However, fast estimation methods often produce inaccurate predictions, while accurate methods are computationally expensive. For example, performance estimation is fast, but its optimistic result is not trusted, which leads to an increased number of iterations in the entire chip design process; routability estimation is typically expensive and not directly integrated into placement. To address these challenges, researchers have turned to ML algorithms, which primarily focus on either accuracy or speed, or rarely both.

3.2 Regression machine learning algorithms

Design result estimation problems are commonly formalized as regression problems, where inputs are mapped to outputs containing at least one continuous value. Roughly, for regression problems, a training set with N samples is given, $(\boldsymbol{x}_i, y_i)$ for $0 \leqslant i < N$, then $y_i* = f(\boldsymbol{x}_i, \boldsymbol{\theta})$ is trained with these N samples, where the error $\mathcal{E}(\boldsymbol{y}, \boldsymbol{y}*)$ is minimized between the given samples and the results from the regression function (i.e., regression model) f, and θ is the parameters of regression function f. This process is shown in Figure 3.

Formalized as regression problems, design result estimation problems can be solved with regression ML algorithms, including Gaussian process (Subsection 3.2.1), multivariate adaptive regression splines (Subsection 3.2.2), decision tree & random forest (Subsection 3.2.3), neural networks (Subsection 3.2.4), and ensemble learning (Subsection 3.2.5). Table 3 shows representative related studies in terms of the adopted regression ML algorithms.

3.2.1 *Gaussian process*

Gaussian process regression (i.e., typically used as a default surrogate model in BO) is a process to model the relationship between input and output, i.e., the probability distribution over the regression function $\mathcal{F}$. Essentially, GP assumes that $p(f(\boldsymbol{x}_0), \ldots, f(\boldsymbol{x}_{N-1}))$ is a jointly Gaussian distribution, and the regression function $f \sim \mathcal{N}(\text{mean}(\boldsymbol{X}), \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j))$, where mean$(\cdot)$ is the mean function and $\kappa(\cdot, \cdot)$ is the covariance function of the assumed distribution, respectively. When predicting with GP regression, the

Table 3 Regression machine learning algorithms used to solve the design result estimation problems

Algorithm	Problem	Ref.	Training samples	Fitting function	Error function
GP	Performance estimation	[14]	Not mentioned[a)]	Throughput	MAE
MARS	Routability estimation	[62]	9 layouts	Routing congestion	MAE
DT & RF	Static timing analysis	[38]	Nets of 120 circuits	Net delay/slew	MSE
	Quality estimation	[15]	Over 1300 HLS reports	Resource usage & timing correction	RAE
	Routability estimation	[16]	About 6500 IR operations	Routing congestion	MAE
	IR drop estimation	[39]	96 floorplans	IR drop	Not mentioned[b)]
NNs	Performance estimation	[17]	Cross-platform instances	Speedup	RMSE
	Routability estimation	[43]	More than 300 floorplans	#DRV/DRC hotpots	MSE
EL	Performance estimation	[18]	2700 HLS reports	Throughput & throughput-to-area	RMSE
	Static timing analysis	[27]	1248 DEFs	Slack	MSE

a) Ref. [14] did not mention the type and number of training samples used.
b) Ref. [39] did not mention the error function that they used.

joint distribution still follows Gaussian distribution, i.e., $p(\boldsymbol{y}*|\boldsymbol{X}*, \boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{N}(\mu*, \Sigma*)$, where $(\boldsymbol{X}, \boldsymbol{y})$ is the set of training inputs and outputs, $(\boldsymbol{X}*, \boldsymbol{y}*)$ is the set of testing inputs and outputs, $\mu*$ is the mean function of the joint distribution, and $\Sigma*$ is the covariance function of the joint distribution. Normally, the parameters in the joint distribution (e.g., kernel function) can be determined by maximizing the marginal likelihood.

Performance estimation. To address the accuracy issue in performance estimation, researchers have explored using GP to improve the accuracy of estimating throughput on field programmable gate arrays (FPGAs) [14]. Ferianc et al. [14] reused prior knowledge from a standard method of performance estimation on FPGA [93], as the mean function of the GP to anchor the estimation within reliable bounds. They then combine the mean function with collected data to avoid complete reliance on the data while estimating performance. Compared with the standard method, Refs. [14, 93] achieved an approximately 30.7% improvement in accuracy.

3.2.2 *Multivariate adaptive regression splines*

Multivariate adaptive regression splines regression can be taken as an extension of linear models that accounts for nonlinearities and interactions between variables, as well as the effects of interactions, and can be seen as a form of the nonlinear generalized additive model. MARS has a form of $f(\boldsymbol{x}) = \sum_{i=1}^{k} c_i \mathcal{B}_i(\boldsymbol{x})$, where the $\mathcal{B}_i(\cdot)$ are the basis functions, c_i are the coefficients, and x is the high-order input. Each $\mathcal{B}_i(\cdot)$ is a tensor product basis of regression spines (e.g., $(x_i - t_i)$ or $(t_i - x_i)$) to represent vector input x, e.g., $(x_i - t_i)(t_j - x_j)(t_k - x_k)$, where t_i, t_j, t_k are values from input $\boldsymbol{x}$. The MARS model is built through forward passes and backward passes. In the forward pass, MARS multiples $f(\cdot)$ with the regression spines of a new value x_i in $\boldsymbol{x}$ gradually, i.e., $(x_i - t_i)$ and $(t_i - x_i)$. In the backward pass, the MARS model gradually removes basis functions that cause the smallest increase in the error. Until the error stops improving through the greedy search style backward pass, MARS select the best model as the final regression model.

Routability estimation. MARS has been proposed to address the challenge of accurate routability estimation in physical design [62,63]. In particular, Qi et al. [62] used an MARS model to predict detailed routing congestion based on layout data (especially global routes). Concretely, the MARS model takes features extracted from the segment number, the via number, and the pin distribution data as input and then predicts the number of design rule violations (#DRV) and routing resource utilization. The MARS model captures nonlinear functions and interactions from layouts and global routes without requiring time-consuming parameter tuning. The learned MARS model is integrated into the global routing step to reduce detailed routing runtime and memory usage while improving solution quality. Compared with a previous analytical model [94], Ref. [62] achieved 34% fewer DRVs and 37% less runtime. Zhou et al. [63] further improved the MARS model by incorporating additional input features such as pin density, routing blockage, and features of global routes and local nets. The trained MARS model predicts the DRVs after routing with an average accuracy of 79.8%.

3.2.3 *Decision tree and random forest*

A decision tree is a supervised learning algorithm that recursively partitions the input space into regions where each region has a local model. DT follows the formula $f(\boldsymbol{x}) = \sum_{m=1}^{M} \omega_m \phi((\boldsymbol{x}), \boldsymbol{v}_m)$, where ω_m is the mean response, and $\phi(\cdot)$ is the basis function in the m-th space. DT normally uses a greedy method to find the optimal partition of the input space, where a full tree is built and then pruned. Random forest further improves the accuracy by leveraging multiple trees, i.e., $f(\boldsymbol{x}) = \frac{1}{T} \sum_{t=1}^{T} f_t(\boldsymbol{x})$, where $f_t(\boldsymbol{x})$ is the t-th tree trained with a randomly chosen subset of inputs.

Static timing analysis. DT & RF have been applied in STA to reduce pessimism and speed up timing closure. Barboza et al. [38] proposed a method for predicting the net delay and slew using an RF model to minimize the pessimism in slack estimation. The RF model takes as input various features extracted from nets, including the driver and sink capacitance, the distance between the driver and the target sink, the max driver input slew, and the context sink locations. The predicted net delay and slew obtained from the RF model are then employed to measure the slack using a program evaluation and review technique (PERT) algorithm [95]. The RF model achieves a receiver operating characteristic (ROC) score of 0.97, indicating that it has low pessimism in the slack estimation compared with a commercial tool (0.85) and other techniques, such as Lasso and MLP.

Quality estimation. DT & RF have been investigated for solving the problem of quality estimation, which involves measuring the hardware characteristics of a design, such as resource usage (especially area), timing, and power. Dai et al. [15] proposed a method that utilizes several ML models, including linear regression, NN, and XGBoost [96], as regression models for estimating the resource usage (i.e., LUT, FF, DSP, and BRAM) and timing correctness of a design by analyzing HLS reports. The input data of these regression models consist of 87 selected features, which are obtained by removing redundant and irrelevant features from a set of 234 features. Experimental results demonstrate that XGBoost (1) is the most competitive ML model among the regression models and (2) reduces the relative absolute errors (RAEs) of the resource usage and timing correct estimations by up to 138%, compared with HLS reports generated by a commercial tool.

Routability estimation. DT & RF have been applied to address the problem of routability estimation, which aims to reduce congestion prediction errors in both the logic design [16] and physical design [25] stages. In the logic design stage, Zhao et al. [16] proposed a gradient-boosted regression tree (GBRT) model for predicting routing congestion in FPGA HLS and identifying highly congested regions in HDL source codes. They extract 302 possible features from operator information, dependency among operators, and scheduling and global information as input to the GBRT model. Experimental results demonstrate that the GBRT model achieves the smallest errors compared with linear regression and NNs, with errors of 6.71% and 10.05% for vertical and horizontal routing congestion, respectively. In the physical design stage, Cheng et al. [25] proposed a DT regression model and a boosted DT regression model for predicting half-perimeter wirelength (HPWL) and routing congestion, respectively. These two models outperform linear regression, NNs, and Poisson regression on average in the two tasks.

IR drop estimation. Ho et al. [39] proposed IncPIRD, an XGBoost-based model that predicts the static IR drop based on the prediction before the modification. The input data of the XGBoost model in IncPIRD are features extracted from technology files (e.g., DEFs and LEFs) using Kirchhoff's circuit law (KCL), Kirchhoff's voltage law (KVL), branch equations, and superposition. IncPIRD achieves a 22–1000× speedup compared with RedHawk[1)], a golden IR drop signoff tool, with an average error of less than 1 mV. Moreover, IncPIRD exhibits strong generalizability that can handle the modification of macro blocks, standard cell blocks, the PDN structure, and power pads without requiring retraining the model.

3.2.4 *Neural networks*

Neural networks, including deep neural networks (DNNs), differ from traditional ML algorithms by having a multi-layer architecture where each layer contains multiple neurons. These neurons are connected to the next layer with synapses, usually with weights, in different ways, such as connecting all neurons (fully-connected layer) or partial neurons (convolutional layer). The output of each neuron is computed by a non-linear function known as the activation function. Combining different types of layers, various NN models can be built, such as the multi-layer perceptron (MLP), the convolutional neural network (CNN),

1) Ansys. RedHawk user guide. https://www.ansys.com/.

the Transformer, the graph neural network (GNN), and the generative adversarial network (GAN). NNs are trained using backpropagation, where the error between network outputs and desired outputs (labels) is propagated from the final layer to the first layer while modifying synapse weights. During prediction, input flows through the network layer by layer, and the final layer acts as a linear regression model. Although there are NN methods that do not rely on backpropagation [97–100], they have not yet been applied to chip design.

Performance estimation. Makrani et al. [17] proposed a performance estimation tool called XPPE, which employs an MLP to predict the potential throughput speedup of porting an application from one FPGA device to another. XPPE extracts features from HLS reports, the target FPGA's available resources, and the application for cross-platform, and uses a 3-layer MLP to predict the throughput speedup of the target FPGA. The MLP takes the extracted features as input and uses root mean squared error (RMSE) as the loss function for training. Experimental results demonstrate that the coefficient of determination between the estimated speedup and the actual speedup is greater than 0.97.

Static timing analysis. NNs have been utilized in STA to predict the path delay and arrival time in both logic design [19] and physical design [40, 41] stages. In the logic design stage, De et al. [19] proposed a GNN-based hybrid model to improve circuit delay prediction accuracy. First, they extract critical HLS combinatorial paths from HLS timing reports and translate them into dataflow graphs. Then, HLS path features are classified into global features (e.g., component counts and cumulative HLS delays) and local features (e.g., structural connectivity and component features). Next, the local features are used to generate graph embeddings through a GNN. Finally, a non-graph regression model (e.g., RF) is utilized to predict the circuit delay using both the global features and the graph embeddings. The hybrid model improves the delay prediction accuracy by 93% compared with simple additive models. In the physical design stage, Guo et al. [40] utilized a GNN model to improve arrival time prediction accuracy. First, the given circuit is mapped to a heterogeneous graph where the edges represent nets and the cells/nodes represent pins. Then, a GNN model with 3 convolution layers is employed to predict the net delay. The GNN model takes physical information of pins, cells, and nets as input features. Concretely, each layer performs graph broadcast and reduction along net edges and reversed net edges, respectively. Next, an MLP with 3 layers is used to predict the cell delay. Finally, the arrival time is computed level-by-level, where the cell delay is predicted by the MLP, and the net delay is predicted by the GNN model. Compared with the previous state-of-the-art (SOTA) model which is based on RF [38], Ref. [40] achieved higher accuracy on the test set. Additionally, Yang et al. [41] developed a transformer-based model to estimate the path delay. First, the circuit is divided into timing paths. Then, each path's physical and timing features are treated as sequential data and input into a transformer [11] to predict the residual value of the path delay. Compared with the RF-based model described in [38], the prediction error of the path delay is less than 3.12% for unseen circuits in terms of relative root mean squared error (rRMSE).

Routability estimation. In the routability estimation problem, NNs, including MLPs [42], CNNs [26, 43–45], and GNNs [46, 47], have been widely used to improve the accuracy and efficiency of routing congestion prediction. For the CNN-based work, Xie et al. [43] proposed RouteNet to predict #DRV and the locations of design rule check (DRC) hotpots. RouteNet uses an 18-layer ResNet [10] to predict the probability of different #DRV classes based on three types of features extracted from macros and a pre-routing congestion estimator using rectangular uniform wire density (RUDY) [101]. The results show that RouteNet achieves similar accuracy with less runtime compared with a global router in Cadence[2)]. In addition, RouteNet detects DRC hotpots before detailed routing using a 9-layer fully convolutional network (FCN), consisting of 7 convolutional layers and 2 transposed-convolutional layers. Besides the features used in the former ResNet model, the FCN model is additionally fed with features extracted from detailed placement layouts and global routing layouts. The result of the DRC hotpot location detection achieves a great progress of 50% accuracy improvement compared with Cadence. Besides, the result is also better than the support vector machine (SVM) and logistic regression-based prediction. As CNNs have been proven to be good at image recognition while layout information (e.g., pin density) can be treated as images, there are also some other CNN-based routability estimation studies [26, 44, 45] that follow [43]. Liang et al. [44] proposed J-Net, a general DRC hotpot prediction framework before global routing, which uses a customized architecture based on U-Net [102]. The model takes input channels of different resolutions and feeds them to different levels at the encoding path to address the mixed input resolution issue, resulting in a 37%, 40%, and 14% improvement in true positive rate (TPR) compared

2) Cadence. Cadence encounter user guide [cited Nov. 3, 2022]. http://www.cadence.com.

with the FCN, conditional GAN (cGAN), and CNN, respectively. Huang et al. [26] used a CNN-based model to predict #DRV which is embedded into a simulated annealing (SA)-based macro placer to optimize floorplan. The CNN model takes macro density maps, pin density maps, and connectivity maps as three kinds of input features, and the optimized floorplan reduces violations compared with the original floorplan. Liu et al. [45] proposed an FCN-based model that predicts congestion hotspots, incorporated into DREAMPlace [58] to improve design routability. The prediction model takes three features (i.e., RUDY, pin density map using RUDY, and macro region) from the global placement solution as input. Experimental results show that Ref. [45] achieved a 9.05% reduction in congestion rate and 5.30% reduction in wirelength compared with DREAMPlace [58] and NTUplace4dr [103], respectively. For the GNN-based work, Ghose et al. [46] proposed an embedding-enhanced GNN that uses matrix factorization methods to learn node embeddings that can be generalized across netlist graphs, resulting in over 90% runtime savings of congestion estimation compared with current methods [104–106]. After that, Wang et al. [47] treated the congestion spot density estimation problem as a pixel-wise classification problem and proposed a heterogeneous GNN architecture named lattice hypergraph neural network (LHNN) to address it. LHNN takes the graph-form representation of circuits, lattice hypergraph (LH-graph), as input and estimates congestion with routing demand regression. LHNN achieves more than 35% improvement in F1 score compared with other pixel-wise classification networks such as Pix2Pix [107] and U-Net [102]. For the MLP-based work, Tabrizi et al. [42] proposed a 1-layer MLP (with 20 nodes) that predicts short violations before routing. The MLP takes features extracted from placed layouts divided into non-overlapping rectangular tiles as input, achieving 90% accuracy for short violation prediction with 7% false alarms.

IR drop estimation. NNs have been used to improve the accuracy and efficiency of IR drop estimation. Specifically, CNN-based methods such as PowerNet [48] and cGAN-based methods like GridNet [49] have been proposed to achieve these goals. PowerNet, proposed by Xie et al. [48], is a CNN-based method that incorporates design-dependent features (e.g., cell locations and timing information) into power maps and predicts the maximum transient IR drop. The method improves accuracy in vectorless IR drop estimation by 9% compared with XGBoost-based methods [108]. GridNet, proposed by Zhou et al. [49], is a cGAN-based framework that accelerates incremental EM-induced IR drop estimation and IR drop violation fixing during PDN synthesis. GridNet takes continuous time and given electrical features as input, and predicts the EM-induced time-varying voltage of the PDN represented as data series images. Experimental results demonstrate that GridNet achieves a $10^5\times$ speedup over the current coupled EM and IR drop estimation tool, EMSpice [109].

Signal integrity analysis. NNs have been applied to accelerate the prediction of EH and EW, which are important parameters in SI analysis [76, 77]. Ambasana et al. [76] used a 2-layer MLP to predict EH and EW based on frequency domain S-parameter data (i.e., return losses and insertion losses), resulting in significant runtime reduction compared with full-factorial time domain analysis. After that, Lu et al. [77] used a 7-layer MLP to predict EH and EW at early design stages through extrapolation with saved coefficients without the need for complex simulations.

Clock power estimation, coupling effect prediction, and net length prediction. NNs have also been applied to estimate clock power [78], coupling effect [79], and net lengths [80]. For clock power estimation, Kwon et al. [78] used multiple MLPs to predict the clock tree components, including clock gating cells, buffers, and their wireloads, which are subsequently used to calculate the clock network power. The MLPs take as input features extracted from CTS constraints and the chip area. Experimental results show that the average error in clock tree component estimation is 13%, while the error in the estimated clock power waveform is only 2%. For coupling effect prediction, Lee et al. [79] addressed the problem as an image-to-image translation problem and utilized a GAN after coupling-free global placement. The GAN takes in features extracted from cell position maps, RUDY maps, and macro position maps as input. Experimental results show that the GAN achieves 91% similarity to the ground truth image with a $50\times$ speedup over NCTUgr [110], a traditional global router. For net length prediction, Xie et al. [80] proposed a graph attention network named Net^2, which models a layout as a graph by treating the nets in the layout as the nodes. Net^2 contains multiple convolutional layers to incorporate edge and node features. Experimental results show that Net^2 improves the accuracy in predicting long nets and critical paths compared with the previous studies [111–113].

3.2.5 *Ensemble learning*

Ensemble learning learns a combination of multiple models to achieve better prediction results than a single model. It can be formed as $f(\boldsymbol{y}) = \sum_{m=1}^{M} \omega_m f_m(\boldsymbol{x})$, where ω_m is the tuning parameter, $f_m(\cdot)$ is a regression model, and $f(\cdot)$ is the final regression function with $\boldsymbol{x}$ as input and $\boldsymbol{y}$ as output. EL can also be taken as a committee method, as each of the regression models gets a weighted vote for the final result.

Performance estimation. Mohammadi et al. [18] proposed an EL model named Pyramid to estimate the performance of an HLS design, specifically the throughput and the throughput-to-area ratio. The Pyramid model is a stack regression framework consisting of up to 50 layers. Each layer involves adding a new base model, which can be a linear regression, an MLP, an SVM, or an RF. The new model is combined with the previous one to estimate either the throughput or the throughput-to-area ratio. The process of adding a new layer is repeated until the maximum number of layers is reached or the RMSE meets the requirement. The input of each base model consists of 72 features, selected from a total of 183 features extracted from HLS reports using linear regression. Experimental results show that the Pyramid model improves throughput accuracy and the throughput-to-area ratio estimation to 95%, which outperforms linear regression, MLP, SVM, and RF.

Static timing analysis. EL has been applied to solve STA in both logic design [20] and physical design [27, 28] stages. In the logic design stage, Que et al. [20] stacked several classification models, including linear regression, DTs, RFs, SVMs, and NNs, to improve the accuracy of STA and speedup timing closure by reducing the number of iterations between FPGA HLS and STA. This work achieves an average area under curve (AUC) score of 0.05 and delivers a 2.7× reduction in iteration counts. In the physical design stage, Chan et al. [27] proposed an SVM kernel boosting method to predict timing failure risk in floorplanning. The key contribution of [27] is proposing a boosting model that uses a weak SVM learner [114] to combine both linear and nonlinear ML techniques. Concretely, the linear technique used is LASSO regression, while the nonlinear techniques include SVM with a radial basis function kernel [115] and NN [115]. They select 27 model parameters from the netlist structure, floorplan context, and layout constraints as the input of the boosting model. This model achieves an average error of less than 10 ps in slack prediction, which is better than the 42 ps achieved by the nonlinear SVM model. Afterward, Zhang et al. [28] applied a stacking model to predict static random-access memory (SRAM) post-layout slack during floorplanning. They first select 27 possible features from design rules, the netlist, and floorplan contexts and then use XGBoost [96] to reduce the possible features to 15 for input into the stacking model. Experimental results show that the stacking model achieves an MAE of less than 23.03 ps in slack prediction, which is superior to typical EDA tools.

Bump inductance prediction and net length prediction. EL has also been used to solve bump inductance prediction [50] and net length prediction [51]. For bump inductance prediction, Cao et al. [50] proposed a piece-wise-linear hybrid surrogate model that combines the prediction from an NN, an SVM, and MARS to make a final prediction. This model achieves an average error of 21.2% or less with only pin maps and technology information. With layout information, the error is reduced to 17.5%. For net length prediction, Hyun et al. [51] proposed an EL algorithm to predict the net lengths from placement-aware synthesis. The algorithm extracts parameters from a given virtual path and compiles them into a handful of parameters through linear discriminant analysis. These parameters are then submitted to 14 ML models, such as LASSO, NNs, and SVMs. The final prediction of net lengths is produced by a weighted sum of the predictions from these 14 ML models, where the weights are determined by the population of the neighbors in parameter space. This algorithm achieves a 93% accuracy, which is better than the 79% accuracy achieved by conventional virtual placement.

3.2.6 *Summary*

It can be summarized that GP, MARS, DT & RF, NNs, and EL are suitable for different design result estimation problems due to their respective strengths and weaknesses.

GP can be applied to design result estimation problems with sufficient human experience and low-dimensional input. On the one hand, GP can incorporate the established analytical foundations as prior knowledge through the specification of a mean function. Based on the established analytic foundations, GP can provide a high-accuracy prediction in an empirical confidence region. On the other hand, the high computational complexity of GP makes it ineffective in high-dimensional space. Therefore, alternative ML algorithms are often necessary to address high-dimensional problems.

Table 4 Design optimization and correction problems in chip design

	Design optimization and correction	Problem description	Input	Output
Logic design	HLS design space exploration	Optimize RTL designs	RTL designs	Optimized RTL designs
Circuit design	Logic optimization	Simplify Boolean expressions	Logic circuits	Optimized logic circuits
Physical design	Floorplanning optimization	Optimize floorplans	Floorplans	Optimized floorplans
	Gate sizing	Choose standard cells	Placed layouts	Optimized placed layouts
	Detailed routing	Correct routing violations	Routed layouts	Corrected routed layouts
	Layout optimization	Optimize layouts	Layouts	Optimized layouts
Verification & test	Test point insertion	Insert test points	Circuits	Circuit with test points
	Test metric optimization	Improve test metrics	Circuits	Tested circuits

MARS is suitable for design result estimation problems where input variables display near-additivity or involve interactions, primarily due to its ability to accurately model relationships in these scenarios by automatically detecting non-linearities and interactions between variables. Additionally, MARS provides strong interpretability and low runtime. However, the susceptibility of MARS to outliers may result in overfitting, which should be taken into consideration while using MARS in practical applications.

DT & RF are well-suited for design result estimation problems that require an interpretable and easily visualized decision-making process, as well as problems that deal with both categorical and numerical data. The primary reason is that the tree-based structure of these models enables a clear and intuitive representation of the decision-making process, which can be easily interpreted and communicated to non-experts. However, DT is prone to overfitting when dealing with data that are either noisy or high-dimensional. Moreover, RF cannot be effective in handling high-dimensional and sparse data.

NNs are adaptable to a wide range of design result estimation problems due to the various types of NN models. For example, GNNs have been successfully applied to problems where circuit netlists can be naturally represented as graphs [19,40]; CNNs can learn more abstract patterns from circuits that can be treated as images [43]; and Transformers are used for timing path analysis since the timing paths can be treated as sequences [41]. However, NNs exhibit limited interpretability due to their complex structure. Furthermore, parameter tuning for neural works is significantly challenging due to the vast number of parameters.

EL is suitable for design result estimation problems where the data is noisy and the predictions made by individual models are subject to a high degree of uncertainty. The main reason is that EL can reduce the risk of overfitting by combining multiple models. Additionally, EL can save considerable time in parameter tuning because EL typically requires only a small number of parameters. However, combining multiple models can lead to limited interpretability, which may hinder the comprehension of underlying data.

4 Design optimization and correction with search machine learning algorithms

4.1 Design optimization and correction problems

Design optimization and correction refer to the problems that optimize design quality and correct design errors. These problems read the design and correct/optimize it without altering its form. Design optimization and correction in different design stages have various purposes, as shown in Table 4. In the logic design stage, design optimization and correction include HLS DSE. In the circuit design stage, design optimization and correction include logic optimization. In the physical design stage, design optimization and correction include floorplanning optimization, gate sizing (adjustment), detailed routing, and layout optimization. In verification and test, design optimization and correction include test point insertion (TPI) and test metric optimization.

HLS design space exploration. High-level synthesis design space exploration searches the input

space or the intermediate representation space of HLS tools, aiming to optimize the design in terms of PPA. In practice, HLS DSE can improve the quality of C/C++ code, which can further improve the quality of the RTL design, netlist, layout, and final chip. However, finding the optimal design is challenging because the quality of the design is largely determined by physical design.

Logic optimization. Logic optimization simplifies the Boolean expression of a circuit before physical mapping to obtain the simplest logical expression possible. The quality of the obtained logical expression significantly impacts the quality of the netlist, layout, and final chip. However, due to highly-complex search space [1, 70], identifying the optimal logic expression remains challenging.

Floorplanning optimization. Floorplanning optimization aims to optimize the floorplans through two approaches. The first approach is searching the neighborhood of a given floorplan for near-optimal or optimal solutions. The second approach is searching for an optimization direction in an existing tool. The goals of floorplanning optimization are twofold: (1) to improve the quality of the design and (2) to accelerate the floorplanning process.

Gate sizing (adjustment). Gate sizing selects the appropriate process, voltage, and temperature (PVT) of standard cells from standard cell libraries, which represent different crafts under specific constraints. It is a critical step in achieving superior PPA and ensuring timing closure and SI satisfaction [57, 60]. Gate sizing can be classified into two groups: adjustment [54, 57] and prediction [60]. Adjustment searches the standard cell libraries to select PVT for cells iteratively. However, the primary challenge with adjustment is its large search space, which requires repeatedly considering massive context details of the netlist. On the other hand, prediction can refer to Subsection 5.1.

Detailed routing. After global routing, detailed routing checks the obtained layout and eliminates any possible congestion. Detailed routing can guarantee the correctness of the chip's routing results by producing exact routes for electric components. However, existing detailed routing approaches have limited generalization ability, which hinders their ability to quickly determine complex design rule constraints, especially when the routing grids become denser in advanced technology nodes [66, 68].

Layout optimization. Layout optimization aims to optimize the placement and routing processes by finding the optimal/near-optimal parameters or guiding the search directions. It is generally divided into two categories: placement optimization and routing optimization. Placement optimization aims to accelerate the placement process [52, 56], while routing optimization aims to improve the quality of routing results and speed up the routing process [64, 65].

Test point insertion. Test point insertion adds control and observation points to a circuit to improve fault coverage while minimizing performance loss. The control points set signal lines to specified logic values, while observation points make nodes observable. TPI is effective in improving circuit testability, resulting in a faster and less complex test. However, two difficulties are associated with TPI. First, achieving the optimal TPI is time-consuming because it is an NP-complete problem [116]. Second, inserting test points can reduce the quality of the area, power, and timing of the circuit [82].

Test metric optimization. Test metric optimization aims to improve the quality metrics of verification and test, such as performance, generalizability, and robustness. Therefore, improving the metrics can lead to more efficient, generalized, and robust verification and test. However, improving the metrics is time-consuming and requires increased computational complexity due to the exponential relationship between the circuit scale and the test space.

Summary. In summary, the design optimization and correction problems introduced above are characterized by the trade-off between design quality and runtime. While these problems aim to achieve high-quality designs quickly, striking a balance between these two factors can be challenging. One example is HLS DSE, where obtaining a high-quality RTL design is time-consuming due to the need to explore a vast design space thoroughly. However, accelerating the DSE process can compromise the quality of the obtained RTL design because less design space can be explored within the limited exploration time. As a result, researchers have turned to ML algorithms to optimize design quality and runtime to better balance this trade-off.

4.2 Search machine learning algorithms

Currently, design optimization and correction problems are commonly formalized as search problems that find the optimal or near-optimal solution from a search space. Roughly, for search problems, a set of states $\{\boldsymbol{y}\}$ is given, known as the search space $\mathcal{F}$. At the start of the search, a start state $\boldsymbol{y}_0$ in the state set is first selected randomly or given by an initialization algorithm. Then, the search space is explored

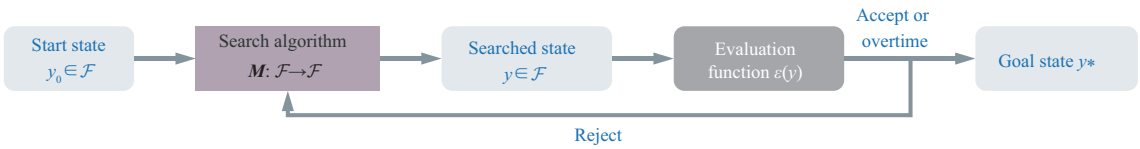

Figure 4 (Color online) Framework of the search algorithm.

Table 5 Search machine learning algorithms used to solve the design optimization and correct problems

Algorithm	Problem	Ref.	States	Evaluation metric(s)
DT & RF	Floorplanning optimization	[29]	Floorplans	PPA
NNs	HLS design space exploration	[23]	RTL designs	QoRs
	Logic optimization	[81]	Logic circuits	Delay & area
	Gate sizing	[54]	Standard cells	Design quality
	Layout optimization	[64]	Routed layouts	Confidence bound
	Test point insertion	[82]	Circuit nodes	F1 score
	Test metric optimization	[83]	Test states	Coverage
RL	HLS design space exploration	[24]	RTL designs	Design quality
	Logic optimization	[85]	Logic circuits	QoRs
	Gate sizing	[57]	Standard cells	TNS
	Detailed routing	[66]	Routed layouts	DRVs
	Test point insertion	[86]	Circuit nodes	Coverage

with an ML search algorithm $\boldsymbol{M} : \mathcal{F} \to \mathcal{F}$, which produces a search state $\boldsymbol{y}$ at each step. The ML search algorithm usually produces the search state by predicting the search direction and step length or learning a heuristic function. An evaluation function $\mathcal{E}(\boldsymbol{y})$, which takes one or more metrics into consideration, is applied to determine whether the current searched state $\boldsymbol{y}$ is acceptable. Finally, the search stops once the evaluation function $\mathcal{E}(\boldsymbol{y})$ shows the current state is acceptable (i.e., a goal state $\boldsymbol{y}*$ is found) or the maximum runtime T is achieved. This process is shown in Figure 4.

Formalized as search problems, design optimization and correction problems can be solved with search ML algorithms, including decision tree and random forest (Subsection 4.2.1), neural networks (Subsection 4.2.2), and reinforcement learning (Subsection 4.2.3). Table 5 shows representative related studies in terms of the adopted search ML algorithms.

4.2.1 *Decision tree and random forest*

HLS design space exploration. Existing studies have explored the use of DT & RF to solve the problem of HLS DSE and reduce the runtime of HLS tools [21, 22]. To accelerate the convergence of the DSE, Liu and Carloni [21] proposed an RF-based model that selects the best knob-setting with binary choices. Additionally, Ref. [21] employed a transductive experimental design method [117] to sample various micro-architecture options, which makes it easier to train the model and improve the efficiency of HLS. Following this trend, Mahapatra and Schafer [22] proposed an efficient DT-based SA algorithm, FSA, which uses a simulated annealer to generate a training dataset and then uses that dataset to construct a DT. Based on the results of the DT, FSA can (1) fix the synthesis directives (i.e., pragmas) that facilitate minimizing/maximizing the cost function objective and (2) continue the annealing procedure with the DT. Experimental results show that FSA improves efficiency by 36% compared with the annealer.

Floorplanning optimization. To improve the PPA quality of floorplans, the use of DT & RF has been proposed for solving the floorplanning optimization problem. Shanthi et al. [29] introduced a four-phase C4.5 DT-based approach to floorplanning. The approach uses a supply voltage assignment for each core in the first phase, followed by island partitioning using a C4.5 DT algorithm [118] in the second phase. In the third phase, a genetic algorithm is used to optimize the blocks for island-level floorplanning. Finally, SA is applied in the fourth phase for chip-level floorplanning, where islands are rotated or swapped. Experimental results show that the DT algorithm achieves 90% accuracy in island partitioning, while the genetic algorithm and SA effectively decrease the costs and runtime of floorplanning.

4.2.2 *Neural networks*

HLS design space exploration. Kwon and Carloni [23] proposed an NN-based model for hybrid-sharing multi-domain transfer learning to improve exploration efficiency in HLS DSE. The model extracts knowledge from spaces to be explored and applies the extracted knowledge in the early stage of the DSE. The approach includes a shared model connected with an independent sub-model for each task to reflect the diversity of various source and target applications. The experimental results in [23] show that the NN-based model can effectively reduce the runtime and cost of HLS DSE.

Logic optimization. Neto et al. [81] proposed an automated hybrid logic optimization method named LSOracle to improve the area-delay product. The key approach of LSOracle is to combine two SOTA logic optimizers, an and-inverter graph (AIG) and a majority-inverter graph (MIG), and use an MLP to select the optimal optimizer for different portions of circuits automatically. These portions are divided by k-way partitioning. Experimental results demonstrate that LSOracle improves the area-delay product by 6.87% and 2.70% on average compared with AIG and MIC, respectively. Furthermore, LSOracle achieves a small area similar to AIG and a high speed comparable to MIG.

Gate sizing (adjustment). Zhou et al. [54] have explored the use of a heterogeneous GNN to accelerate gate sizing adjustment. Their approach aims to speed up the widely-used Lagrangian relaxation algorithm by narrowing down the range of cells. Concretely, they first represent the timing graph as a heterogeneous directed graph and then design a heterogeneous GNN as the encoder. The GNN is trained to imitate the selection behavior of the Lagrangian relaxation algorithm using imitation learning and then used to predict which cells need to be changed. Experimental results demonstrate that the proposed approach achieves a 22.5% reduction in runtime while maintaining comparable design quality when compared with the non-ML-based method [119].

Layout optimization. NNs have been explored to solve layout optimization problems in the context of placement [52, 53] and routing [64, 65] optimization. For placement optimization, Lu et al. [52] proposed PL-GNN, a two-stage GNN-based framework. In the first stage, GNNs are used for unsupervised node representation learning of a given netlist to learn accurate node representations associated with the logical affinity and attributes of the netlist. In the second stage, instances are divided into clusters that serve as placement guidance for commercial tools to optimize metrics. Compared with the placement process in Synopsys IC Compiler II, the proposed framework reduces the runtime by 85.7% while maintaining comparable wirelength and power. After that, Guo and Lin [53] proposed a timing-driven placement paradigm based on a differentiable timing engine inspired by an MLP. In this paradigm, the STA is mapped to the MLP propagation, allowing the MLP training to optimize the design timing. Experimental results demonstrate that the proposed paradigm can reduce total negative slack (TNS) and worst negative slack (WNS) by 32.7% and 59.1%, respectively. For routing optimization, He and Bao [64] proposed a CNN-based depth-first search (DFS) algorithm to improve the efficiency of Monte Carlo tree search [120]. In the search process, the CNN is trained to prioritize available nodes (i.e., routing resources) for a given net. Then, instead of random node selection in the vanilla Monte Carlo tree search, a DFS algorithm is used to select nodes step by step. The CNN-based DFS algorithm accelerates the search process significantly and achieves results of higher quality compared with the vanilla Monte Carlo tree search. Meanwhile, Chen et al. [65] proposed a routability optimization tool, PROS, to improve the design quality of SOTA commercial EDA tools. PROS consists of two key components: an FCN-based predictor and a parameter optimizer. The FCN-based predictor estimates global routing congestion using placement results data, and the parameter optimizer adjusts global routing cost parameters based on the estimation results. The combination of the two components generates better-routed layouts with fewer DRC violations. Experimental results show that PROS reduces DRC violations by an average of 11.65%.

Test point insertion. Ma et al. [82] proposed a graph convolutional network (GCN)-based model for improving fault coverage and reducing the number of inserted test points in logic circuits' irregular graph representations. The proposed GCN-based model consists of three key stages, an aggregator, an encoder, and a classifier. These generate node embedding, predict difficult-to-observe observation point candidates, and propose observation point insertion, respectively. The trained GCN-based model achieves superior accuracy in predicting difficult-to-observation nodes compared with traditional ML models, such as linear regression, RF, SVM, and MLP. Moreover, it reduces observation points by 11% and test pattern count by 6% with similar fault coverage compared with a commercial testability analysis tool [82].

Test metrics optimization. Gad et al. [83] proposed a two-phase graph-based framework with an adaptive NN to improve simulation-based verification in sequential circuits. In the first phase, the

adaptive NN receives test states and instructions to generate a graph, where the nodes represent the states of a design under test and the edges represent operations between the states. In the second phase, a test instruction sequence is extracted from the graph using the Dijkstra algorithm, a shortest-path algorithm. Experimental results show that the proposed graph-based framework achieves full coverage closure using various training datasets. In contrast, other methods' quality deteriorates with a decrease in the training dataset size. In short, the framework overcomes the sensitivity of RF and fixed NN to the size and quality of the training dataset [83].

4.2.3 *Reinforcement learning*

Reinforcement learning is an ML framework that trains an intelligent agent to take action in an environment where the agent receives a reward instead of being told to take which action. Typically, reinforcement learning is modeled as a Markov decision process (MDP), including (1) a set of environment and agent states $\mathcal{S}$, (2) a set of available actions $\mathcal{A}$ of the agent, (3) probability of transition $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ from state s to state s' at time t under action a, and (4) immediate reward $R_a(s, s')$ after transition from s to s' with action a. Based on the MDP, the agent first receives the current state s_t and the reward r_t at each time t. Then, the agent chooses an action a from the available action set $\mathcal{A}$ and sends it to the environment. With the action, the environment moves to a new state s_{t+1}. Meanwhile, a reward r_{t+1} related to the transition $s_t \xrightarrow{a} s_{t+1}$ is determined. The goal of the agent is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ (i.e., $\pi(a, s) = \Pr(a_t = a | s_t = t)$), which maximizes the expected cumulative reward.

HLS design space exploration. Wu et al. [24] proposed an RL-based framework named IronMan to improve the RTL design quality. The framework comprises three key components: GPP, RLMD, and CT. Firstly, GPP is a GNN-based performance predictor that estimates resource utilization and critical path timing for HLS designs. The GNN-based predictor not only suits the data form (i.e., design flow graphs) but also demonstrates faster than Vivado HLS[3)]. Secondly, RLMD is a deep reinforcement learning (DRL)-based multi-objective DSE engine that finds the optimal resource allocation strategy. Lastly, CT is a code transformer that extracts data flow graphs from the intermediate representations generated by Vivado HLS and produces synthesizable C/C++ programs with directives optimized by RLMD. Experimental results show that (1) GPP reduces prediction errors by 10.9× in resource usage and 5.7× in timing when compared with Vivado HLS; (2) RLMD reduces resource usage by 12.7% and 12.9% compared with genetic algorithm and SA, respectively; and (3) IronMan uses 2.54× fewer DSPs and attains 6× shorter latencies than Vivado HLS.

Logic optimization. Several RL-based studies have been explored to solve the problem of logic optimization with the goal of improving the quality of output circuits [70, 84, 85]. Haaswijk et al. [84] first modeled the logic optimization problem as a deterministic MDP, and then used DRL to learn how to navigate the MDP. Concretely, they model the policy function of the DRL as a GCN-based NN, which is trained to predict the most promising candidate action. Experimental results demonstrate that the proposed approach can deliver an 86% improvement in the logic depth of realistic circuits compared with the SOTA method, resyn2. Meanwhile, Zhu et al. [85] also modeled the logic optimization problem as an MDP and used a trained GCN-based policy network to navigate the MDP. Instead of using MIGs in [84], they use AIGs to represent the logic expressions, thus achieving greater scalability for more experiments. Experimental results show that Ref. [85] achieved faster speed using the same action space compared with the SOTA, resyn2. In contrast to the GCN-based policy network, Hosny et al. [70] proposed DRiLLS, which trains an advantage actor-critic (A2C) agent to minimize area subject to timing constraints. DRiLLS (1) maps the search space to a "game" and (2) applies the A2C agent to maximize its reward (i.e., reduce area subject to timing constraints) by iteratively selecting primitive transformations with the highest expected reward. As experimental results show, DRiLLS improves QoRs by 13% on average, which is better than [121, 122].

Floorplan optimization. He et al. [30] proposed an RL-based approach to obtain desirable floorplans by using a local search algorithm. The primary objective of the work is to develop algorithms for solving combinatorial problems without human knowledge. The authors first formulate the local search problem as an MDP and then use a deep Q-learning [13] algorithm to train the agent to select a candidate neighbor solution. Experimental results show that the proposed RL-based local search outperforms SA and can

3) Xilinx. Xilinx Vivado high-level synthesis. 2021 [cited Nov. 4, 2022]. https://www.xilinx.com/products/design-tools/vivado.html.

effectively reduce the dead space in floorplanning.

Gate sizing (adjustment). Lu et al. [57] proposed an autonomous gate sizing agent, RL-Sizer, for timing optimization. In RL-Sizer, the gate sizing problem is modeled as an MDP, and GNN-based RL algorithms are used to solve it. The input features of RL-Sizer can be obtained from timing-related instance characteristics using a GNN-based encoder. The reward function of RL-Sizer is defined as the TNS change of a local graph, which threads across various instances and accelerates the learning process. Experimental results show that RL-Sizer achieves comparable TNS optimization on six commercial designs that use advanced technology nodes, compared with Synopsys IC-Compiler II.

Detailed routing. Solving detailed routing by RL has been explored to enhance solution generalizability and quality and reduce runtime [66–68]. Liao et al. [66] proposed the attention router, an RL-based router that employs an attention model to tackle the track-assignment detailed routing problem. In the attention router, the detailed routing algorithm first encodes design rules and constraints. An attention-model-based algorithm called REINFORCE is then applied to sequence device pairs that need to be routed, which is the most critical step in detailed routing. Experimental results show that the attention router not only demonstrates generalizability to unseen problems but also achieves a more than 100× speedup over a novel genetic router with similar solution quality. Several following studies are inspired by [66], including [67, 68]. Ju et al. [67] proposed a multi-agent RL-based detailed router to eliminate routing congestion, which performs 3 steps as follows: (1) routing asynchronization by modeling nets as agents and regarding pin-connection tasks as path planning, (2) assigning each agent a local view field to reduce feature size and training difficulty, and (3) setting up an information storage unit for each agent's communication to eliminate routing congestion. Experimental results show that the router improves the success rate of detailed routing to 96% and reduces the cost by 15.7%, compared with the baselines [123, 124]. Meanwhile, Lin et al. [68] proposed an asynchronous RL-based framework and a transfer learning algorithm to solve the detailed routing problem. The RL-based framework explores optimal ordering strategies, while the transfer learning algorithm accelerates the detailed routing speed. Compared with the SOTA router [125], Ref. [68] could reduce 26% of DRC violations.

Layout optimization. Vashisht et al. [55] proposed a placement framework that combines cyclic RL and SA to improve the quality of the placement solution. The placement problem is formulated as a sequence pair and solved in a customized environment. The proposed framework leverages the strengths of both RL and SA to provide a good initial state for SA. Experimental results demonstrate that the framework improves the quality of the placement solution by providing better SA initialization. After [55], Agnesina et al. [56] proposed a DRL framework for optimizing placement. In the training stage, an autonomous agent is trained using RL algorithms that perform self-search. The agent can learn to optimize parameters without human participation, which helps to overcome the data sparsity and reduce the placement runtime. In the testing stage, handcrafted features from graph topology theory and graph embeddings generated by unsupervised GNNs are leveraged to optimize placement parameters for unseen designs. Experimental results show that the trained RL agent can deliver an improvement of 11% and 2.5% in wirelength compared with manual work and a SOTA tool [126], respectively, on unseen netlists.

Test point insertion. Shi et al. [86] proposed a DRL-based TPI approach, DeepTPI, to improve fault coverage in logic built-in self-test (LBIST). In this approach, netlists are modeled as directed graphs, and a graph-based value network is proposed to predict the action values. The DRL agent combines a GNN and a deep Q-learning network (DQN) to improve the test coverage. DeepTPI is experimentally shown to outperform pattern simulation [127] in terms of test coverage improvement.

4.2.4 *Summary*

It can be summarized that DT & RF, NNs, and RL can each be utilized to tackle various design optimization and correction problems or to serve distinct functions in the problem-solving process due to their unique strengths and limitations.

DT & RF can serve as aides in tackling design optimization and correction problems by partitioning regions and searching for parameters. The main advantage of using DT & RF in this capacity is that they are typically employed as classification or regression algorithms rather than search algorithms. However, to the best of our knowledge, there is a scarcity of existing work that exclusively employs DT & RF to address design optimization and correction problems alone.

NNs are adaptable to a wide range of design optimization and correction problems due to the various

Table 6 Design construction problems in chip design

	Design construction	Problem description	Input	Output
Logic design	–	–	–	–
Circuit design	Physical mapping	Generate netlists	RTL designs	Netlists
Physical design	Module placement	Place macros/instantiated modules	Netlists	Floorplans
	Global placement	Place standard cells	Floorplans	Placed layouts
	Gate sizing	Choose standard cells	Placed layouts	Cell types
	PDN synthesis	Generate PDNs	Layouts	PDNs
	Clock tree synthesis	Generate clock tree	Placed layouts	Clock trees
	Global routing	Routing	Unrouted layouts	Routed layouts
Verification & test	Testbench generation	Generate testbenches	Circuits	Testbenches

types of NN models. For instance, MLPs are appropriate for design optimization and correction problems involving complex input/output relationships due to their deep architecture with multiple hidden layers [81], while FCNs are well-suited for problems with inputs of different resolutions as they can handle inputs of arbitrary size and produce outputs of the same size [65]. However, the complicated structure of neural networks results in limited interpretability. Additionally, optimizing a vast number of parameters poses a significant challenge for NN parameter tuning.

RL is suitable for design optimization and correction problems, especially when the optimal solution depends on previous actions and environmental responses. The primary reason is that RL models the search problem as an MDP and takes into account the long-term reward when searching. Additionally, the MDP formulation process in RL relies only on the current state without being influenced by the sequence of past states. Moreover, RL can be applied to problems without clear supervision. However, RL is very sensitive to the initial conditions and parameters, so it may not obtain the optimal solution if given unsuitable initial conditions or parameters. More importantly, RL is ineffective in addressing large-scale problems due to the expensive computational cost.

5 Design construction with generation machine learning algorithms

5.1 Design construction problems

Design construction refers to problems that change the chip's representation (e.g., physical mapping) or complete the layout (e.g., global placement). These problems are crucial steps in chip design that analyze the design and generate a netlist, layout, or layout components in different stages for different purposes, as shown in Table 6. In the circuit design stage, design construction includes physical mapping. In the physical design stage, design construction includes module placement, global placement, gate sizing (prediction), PDN synthesis, CTS, and global routing. In verification and test, design construction includes testbench generation.

Physical mapping. Physical mapping transforms the RTL program into a gate-level representation using a specific technology library. Its primary objective is to generate a basic physical circuit, represented as a netlist, without incorporating any physical information. The subsequent processes further complete this netlist. However, determining the optimal solution in physical mapping is challenging due to the uncertainty regarding design quality and the need to achieve satisfactory generalization [71, 128, 129].

Module placement. Module placement, which plays a crucial role in the floorplanning process, divides the chip into blocks and positioning macros and instantiated modules within those blocks. Effective floorplanning is essential for achieving high-quality module placement. However, obtaining a satisfactory floorplan often relies heavily on expert knowledge and necessitates close collaboration between experts from various domains.

Global placement. Global placement determines the optimal placement of standard cells on the layout while considering various constraints to achieve the highest quality and performance. It is a critical step in the overall placement process, as the quality and performance of the global placement directly impact the chip's overall performance and routability. However, global placement is time-consuming, as it places many standard cells, and its complexity grows exponentially. Furthermore, identifying the theoretically optimal placement layout is challenging due to the diverse objectives of routing optimization, such as wirelength, timing, and resource usage.

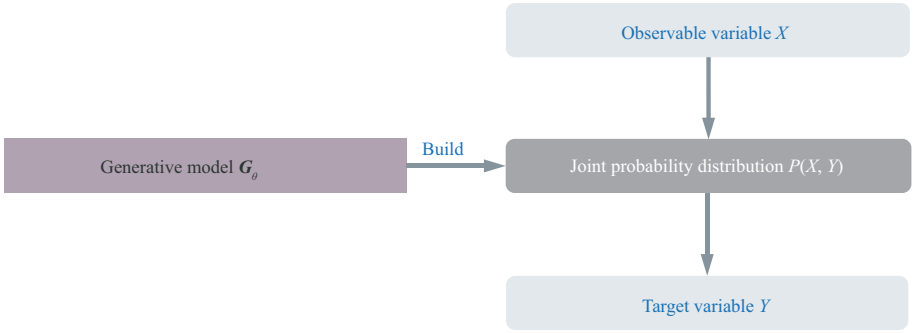

Figure 5 (Color online) Framework of the generation algorithm.

Gate sizing (prediction). Gate sizing prediction, instead of a step-by-step search process in gate sizing adjustment, assigns standard cell types using a generation algorithm. It offers a significant reduction in the runtime compared with the adjustment way, which is introduced in Subsection 4.1. Unfortunately, obtaining the optimal solution for gate sizing prediction remains time-consuming due to the extensive design space involved.

Power delivery network synthesis. Power delivery network synthesis generates a network that facilitates the delivery of power from voltage regulator modules to the circuits on the chip. The PDN plays a crucial role in the overall performance, quality, and reliability of the chip [39, 49]. However, generating an optimal PDN is time-consuming due to the significant number of design iterations involved and the presence of stringent constraints [39, 61].

Clock tree synthesis. See the details in Subsection 2.1.3

Global routing. Global routing allocates routing resources and establishes connections between components, such as macros and standard cells, to generate an approximately acceptable routing scheme. A high-quality global routing result contributes to improved timing and power characteristics. Unfortunately, global routing is excessively time-consuming due to the exponentially large design space and the associated complexity that grows with the number of potential wire configurations.

Testbench generation. Testbench generation generates a comprehensive set of test inputs and their corresponding expected outputs. These inputs and outputs are utilized to verify the functional correctness of a design. A well-designed testbench aims to cover all potential design flaws. However, achieving high coverage often necessitates a large number of test cases, resulting in the presence of redundant test vectors. Consequently, the existence of such redundancy leads to increased runtime during the testing process.

Summary. In summary, speed is a primary concern in the design construction problems introduced above. These problems typically require efficient design runtimes. However, exploring the vast design space and iteratively improving design quality are both time-consuming. For instance, physical mapping explores a design space with more than exponential complexity, particularly as the design scale increases. Similarly, global placement requires iterative improvements to achieve timing closure. To tackle this challenge, researchers have turned to ML algorithms with a focus on improving speed.

5.2 Generation machine learning algorithms

Currently, design construction problems are commonly formalized as generation problems, where inputs are mapped to outputs containing a netlist, layout, or layout components. Roughly, for generation problems, an observable domain $\mathcal{X}$ and a target domain $\mathcal{Y}$ are given. Then, a generative model $\boldsymbol{G}_\theta : \mathcal{X} \to \mathcal{Y}$, where θ is the parameters of the model, is learned to build a joint probability distribution $\boldsymbol{P}(\boldsymbol{X}, \boldsymbol{Y})$ on the given observable domain and target domain. Based on the distribution, observable variable $\boldsymbol{X}$ in the observable domain $\mathcal{X}$ is mapped to target variable $\boldsymbol{Y}$ in the target domain $\mathcal{Y}$. This process is shown in Figure 5.

Formalized as generation problems, design construction problems can be solved with generation ML algorithms, including Bayesian optimization (Subsection 5.2.1), neural networks (Subsection 5.2.2), and reinforcement learning (Subsection 5.2.3). Table 7 shows representative related work in terms of the adopted generation ML algorithms.

Table 7 Generation machine learning algorithms used to solve the design construction problems

Algorithm	Problem	Ref.	Observable variable	Target variable	Evaluation metric(s)
BO	Physical mapping	[71]	RTL designs	Netlists	QoRs
NNs	Module placement	[32]	Netlists	Floorplans	Wirelengths
	Global placement	[58]	Unplaced layouts	Placed layouts	Design quality
	Gate sizing	[60]	Placed layouts	Gate types	Accuracy
	PDN synthesis	[61]	Floorplans & placed layouts	PDNs	Routing resources
	CTS	[88]	Placed layouts	Clock networks	Clock network quality
	Global routing	[69]	Unrouted layouts	Routed layouts	Routability
	Testbench generation	[90]	Circuits	Testbenches	Coverage
RL	Module placement	[34]	Netlists	Floorplans	PPA

5.2.1 *Bayesian optimization*

Bayesian optimization is typically used to find the minimum or the maximum of a function $f(\boldsymbol{x})$ on a bounded set $\mathcal{X}$, which is a subset of $\mathbb{R}^D$. As evaluating the function $f(\boldsymbol{x})$ is generally expensive, BO first constructs a probabilistic surrogate model M for $f(\boldsymbol{x})$. The probabilistic surrogate model is then exploited to select an $\boldsymbol{x} \in \mathcal{X}$ that maximizes or minimizes an acquisition function a (i.e., $\text{argmax}_{\boldsymbol{x}\in\mathcal{X}} a(\boldsymbol{x}, M)$), where the acquisition function balances the exploration and exploitation of the exploration. The selected $\boldsymbol{x}$ is evaluated, and the result of the evaluation is then used to refit the probabilistic surrogate model M. Finally, BO finds the minimum or maximum of the function in a short time with only a few evaluations.

Physical mapping. Grosnit et al. [71] proposed a BO-based generation algorithm named BOiLS, aiming to obtain superior QoRs in the circuit design stage. BOiLS consists of two main steps. Firstly, BOiLS employs specially designed kernels for AIG transformation sequences to fit a surrogate GP model to the QoR data. This enables efficient modeling and prediction of QoRs. Secondly, BOiLS suggests new synthesis flows for evaluation based on the updated GP model. To handle high-dimensional problems, BOiLS utilizes a local trust-region acquisition function maximization technique. Experimental results demonstrate that BOiLS achieves superior QoRs compared with other methods in the majority of the designs tested (8 out of 10). In contrast, the SOTA BO-based method [130] only outperforms BOiLS in one design. Furthermore, BOiLS improves sample efficiency by more than one-third compared with the SOTA method [130].

Module placement. Oh et al. [31] proposed a solution to the macro placement problem by modeling it as a combinatorial optimization problem involving pairs of sequences. They employ BO to address this problem effectively. The main concept is to transfer batch BO on permutations [131] into batch BO on sequence pairs. Moreover, Oh et al. [31] developed a batch acquisition function that suggests multiple candidate solutions for macro placement. This enables parallel evaluation of a batch of candidates, significantly accelerating the BO process. Experimental results demonstrate that the proposed approach in [31] outperforms the SA algorithm in terms of the HPWL metric, indicating better placement quality.

5.2.2 *Neural networks*

Module placement. Researchers have explored the use of NNs to solve the module placement problem, aiming to achieve high-quality floorplans in a shorter runtime [32, 33]. Liu et al. [32] proposed Flora, a graph attention-based floorplanner that efficiently generates floorplans with reduced wirelength. Flora utilizes a graph attention network [132] to capture the connectivity of sub-circuits, and then the mapping between the circuit connectivity and the physical wirelength can be learned. By leveraging this learned mapping, Flora predicts the physical locations of blocks, enabling the generation of high-quality floorplans to guide the module placement process. Experimental results demonstrate that Flora achieves an average reduction of 18% in runtime and 2% in wirelength compared with the SOTA placer, DREAMPlace 3.0 [133]. Building upon Flora, Liu et al. [33] proposed GraphPlanner, a variational GCN-based DL method for floorplanning. GraphPlanner incorporates a clustering method that combines hyperedge coarsening and graph spectral clustering techniques to divide large-scale netlists into clusters with minimized inter-cluster weighted connectivity. Compared with DREAMPlace 3.0 [133], GraphPlanner achieves reductions of 25% in runtime and 4% in wirelength.

Global placement. Researchers have explored the use of NNs to accelerate the global placement process, resulting in several notable studies [58,59]. Lin et al. [58] proposed DREAMPlace, a GPU-accelerated placement framework based on the ePlace/RePlAce family [134, 135]. DREAMPlace transforms the an-

alytical global placement into a NN training problem and employs GPU optimization techniques to accelerate compute-intensive steps such as wirelength and density computation. Experimental results demonstrate that DREAMPlace achieves a speedup of over 30× speedup compared with RePlAce [135] while maintaining similar quality. Building upon DREAMPlace, subsequent versions, including DREAMPlace 2.0 [136], DREAMPlace 3.0 [133], and DREAMPlace 4.0 [137], have been developed to further enhance design quality and reduce runtime for the global placement problem. Liu et al. [59] proposed Xplace, another GPU-accelerated placer that leverages a Fourier NN as a global guide for placement. Xplace achieves a 2× speedup compared with DREAMPlace [58] while delivering improved design quality.

Gate sizing (prediction). Nath et al. [60] proposed a transformer-based generative model for gate sizing, aiming to enhance both accuracy and efficiency. The approach in this work maps the domain of an unoptimized netlist to the domain of the corresponding optimized netlist. Subsequently, a transformer model is employed to train the joint probability distribution between these two domains. The transformer model generates optimal sizes for all gates within a timing path. Experimental results demonstrate that the proposed model achieves a 93% accuracy in gate sizing prediction while offering a substantial speedup of 1400× compared with a commercial EDA tool.

Power delivery network synthesis. Chhabria et al. [61] proposed an NN-based approach to synthesize a PDN based on pre-designed templates. The approach leverages manually-specified templates of PDNs and employs NNs in both the floorplanning and placement stages. In floorplanning, an MLP is utilized to generate an optimized PDN, taking into account estimated current and congestion. In placement, a CNN is employed to optimize the PDN iteratively, considering more accurate current and congestion estimation. The applied NNs in each stage generate a PDN that satisfies IR drop and EM specifications. Experimental results demonstrate that the proposed approach achieves a 3% reduction in routing resources compared with the SA algorithm.

Clock tree synthesis. Lu et al. [88] proposed a novel framework called GAN-CTS, which employs a combination of GANs and RL to tackle the CTS problem. GAN-CTS comprises three key components: a regression model, a parameter generator, and a discriminator. The regression model utilizes ResNet [10] and transfer learning techniques to extract features from placed layouts and predict the CTS outcomes. The parameter generator is the crucial component that employs RL and cGANs to generate optimal CTS parameters. The discriminator is responsible for verifying whether the CTS process satisfies the design rules and specifications. Experimental results demonstrate that GAN-CTS significantly improves the performance of CTS. Specifically, it achieves a 3% average prediction error and reduces clock power by 51.5%, wirelength by 18.5%, and maximum skew by 5.3%, compared with a commercial tool. Furthermore, GAN-CTS attains an F1-score of 0.952 in determining the success of the CTS process. Nonetheless, GAN-CTS incurs additional runtime due to placement and trial routing. To overcome this issue, Koh et al. [89] proposed an NN-based clock tree predictor that employs MLPs to predict the number of clock buffers for each clock branch. These predicted results are used to generate buffer trees and a binary search algorithm is applied to determine the minimum number of buffers. Experimental results show that the proposed approach in [89] can reduce the number of clock buffers by 31% on average compared with a commercial logic synthesis tool.

Global routing. Utyamishev and Partin-Vaisband [69] proposed an NN-based global router that is trained on previously routed layouts and can infer unseen routed layouts. They redefine the global routing problem as an image-to-image processing problem and solve it using a specially designed NN model that includes a variational autoencoder and a custom loss function. The main advantage of their work is the ability to parallelize the global routing process. Experimental results demonstrate that their approach achieves over 5× speedup with comparable routability when compared with the SOTA router, FastRoute [138].

Testbench generation. Zheng et al. [90] proposed NNBNTS, an NN-based configurable framework for test selection, aiming to reduce simulation effort while maintaining similar coverage as random simulation. The key idea behind NNBNTS is to identify the novelty or unexplored part of the test space. To achieve this, NNBNTS takes test features as input and novelty definitions as the output space. The framework has three configurations: (1) an autoencoder that reconstructs the input test vector, where the output space is the reconstructed input space, with high reconstruction errors indicating novelty; (2) a configuration that predicts the correlation between the space of test features and coverage and assigns a novelty score to each test based on NN calculations; and (3) a configuration that generates a non-linear score to evaluate the novelty degree. Experimental results show that NNBNTS reduces simulation time by 53.74% with 99% coverage compared with random test simulation.

5.2.3 *Reinforcement learning*

Module placement. The use of RL to solve the macro placement problem has been extensively explored in order to enhance the quality of floorplans and decrease runtime [34–37]. Mirhoseini et al. [34] proposed an RL-based approach to macro placement, which reduces runtime. They formulate the macro placement problem as an RL problem and train an agent to place macros on a chip canvas. The trained agent sequentially places macros during each iteration of training. In the RL problem, the policy and value networks utilize a neural architecture encoder that predicts rewards for netlists and their placements and produces feature embeddings of the input netlists. Experimental results demonstrate that the proposed method achieves significantly better or comparable PPA in less than 6 h instead of the several weeks required by human experts. Building upon [34], Chang et al. [37] proposed a flexible multiple-objective RL framework to enhance the efficiency and effectiveness of module placement. The framework utilizes a pre-trained model, MOPPO, to support variable weights during inference in objective functions. To accelerate MOPPO training, the framework applies trial macro placements for a suitable cluster size, thereby accelerating reward computation. Experimental results indicate that for two uncorrelated objectives, the framework can recover the Pareto frontier in just a single round of training. In addition, Xu et al. [35] modeled the macro placement problem as an MDP and proposed an end-to-end GCN- and RL-based model to address it. Firstly, they build a GCN to form the graph embedding of netlists. They then apply an A2C algorithm to explore the solution space effectively. Compared with SOTA floorplanners based on heuristic [139, 140][4)], the proposed model offers superior design quality and success rate. However, Ref. [35] did not demonstrate significant improvement in runtime compared with SA. Amini et al. [36] proposed a DRL method to address the floorplanning problem by directly predicting block ids and locations. The method includes three highlights. The first highlight is the utilization of hyper-graph NNs to encode hyper-net information in circuit netlists. Additionally, the method applies transformer-like action selection for large action space exploration, with transferability and generalizability across netlists. This avoids the influence of the number of blocks by fixing the complexity of the parameter space. Moreover, the method trains an RL agent to transfer prior knowledge into new design generation quickly. The proposed method effectively reduces the computing complexity of floorplanning (i.e., from exponential to $O(n)$).

5.2.4 *Summary*

It can be summarized that BO, NNs, and RL are suitable for different design construction problems due to their respective strengths and weaknesses.

BO is appropriate for design construction problems that demand high-quality solutions. This is primarily because BO can dynamically adjust the density of sampling points to explore and exploit the search space more efficiently. Additionally, BO can rapidly identify the global optimal solution by adaptively modifying the location and size of the next sample based on previous sampling points. However, the computational cost of BO is greatly expensive for large-scale data as evaluating the acquisition function for the surrogate model (e.g., GP) can be time-consuming, especially when the function is expensive to compute or the input space is high-dimensional.

NNs possess adaptability to a wide range of design construction problems because of the availability of diverse types of NN models. See the details in Subsections 3.2.6 and 4.2.4.

RL is suitable for design construction problems that can be formulated as combinatorial optimization problems. The primary benefit of utilizing RL in this context is that it is commonly utilized as a search algorithm rather than a generation algorithm. However, since RL is sensitive to initial conditions and parameters, it will miss the optimal solution if the algorithm starts with an unsuitable initialization. Besides, applying RL to solve large-scale problems can be challenging work due to the expensive computation.

6 Future work

Although the application of ML algorithms in chip design has been proven effective in accelerating chip design, reducing design costs, and improving design quality, three key challenges remain yet to be

4) Adya S, Chan H H, Markov I. Parquet 4.5: Fixedoutline floorplanner. 2006 [cited Nov. 3, 2022]. http://vlsicad.eecs.umich.edu/BK/parquet/.

addressed.

(1) Some traditional studies of chip design still depend on human participation in the high-level (Challenge-1). Many critical steps, such as hardware programming and detailed placement, rely heavily on manual design and correction by chip designers. Additionally, chip performance specifications are often written and tuned by human experts based on design quality and rules. This reliance on human expertise can result in expensive development costs and long production cycles.

(2) Existing studies fail to globally optimize design quality (Challenge-2). This is due to the fact that a chip is designed and optimized stage by stage, and the strategies used in later stages are not foreseen during earlier stages. As a result, it is challenging for the existing studies to achieve a globally optimal design.

(3) Existing studies have limitations in terms of practicality (Challenge-3). Currently, these studies' design scale is considerably smaller than real chips, making applying them in industrial chip design processes challenging. Additionally, the community lacks realistic, diverse, and standardized datasets and benchmarks based on actual data from commercial chips. This is a special case for chip design with ML algorithms. Therefore, applying existing studies in the industrial chip design process is difficult, which can be a great hurdle for the community in practice.

To address the aforementioned challenges, several future studies can be explored as follows.

(1) Single-stage end-to-end generation. ML algorithms can be applied to replace each stage of the current chip design (i.e., single-stage end-to-end generation), including HLS, logic synthesis, floorplanning, placement, CTS, and routing (i.e., to address Challenge-1). Specifically, single-stage end-to-end generation takes the input of a chip design stage as the input of the generation and the output of the chip design stage as the output. Since the end-to-end generation does not need iterations between design steps in the stage, it can effectively speedup the design process. Besides, applying ML algorithms in single-stage end-to-end generation can also eliminate human participation in each stage. However, it is still hard to conduct the single-stage end-to-end generation with an acceptable correct rate or high design quality by leveraging ML algorithms. Thus, further exploring in this direction is of great significance but with challenges.

(2) Cross-stage end-to-end generation. Replacing two or more adjacent stages with ML algorithms (i.e., cross-stage end-to-end generation) can be further explored (i.e., to address Challenge-1 and Challenge-2). Currently, such generation is only discussed in [141], where a joint learning framework is proposed to simultaneously replace the placement and routing. The proposed framework can obtain better design quality compared with SOTA single-stage studies [34, 58, 135], which demonstrates great potential for cross-stage design and optimization. Moreover, like single-stage end-to-end generation, cross-stage end-to-end generation can also offer a fast chip design without human participation in the replaced stages. However, the design space in the cross-stage end-to-end generation is much more complicated than that of the single-stage one. The main reason is that a typical cross-stage generation involves multiple stages with different high complexity. Here, the complexity of cross-stage generation is the product of the complexity of the multiple stages, which incurs a highly-complex design space. As a result, conducting an efficient cross-stage end-to-end generation can be much more challenging.

(3) Whole-process end-to-end generation. ML algorithms can be used to read chip specifications and generate a finished layout in an end-to-end way, and thus replace the current chip design process (i.e., whole-process end-to-end generation, to address Challenge-1 and Challenge-2). For the input, chip specifications can be roughly divided into functional specifications and performance specifications. Chip functional specifications can be given in the form of input/output pairs or a (part of) truth table of the expected chip, which is simple work for the users. Chip performance specifications are given based on human experience and the actual design result. For the output, the finished layout is complemented by components and routes and then delivered to the foundry for tape-out. Research on whole-process end-to-end generation has a great potential to reduce the professionalism of chip design because human participation in chip design can be entirely eliminated. Besides, the appliance of end-to-end generation algorithms in whole-process generation can also speedup chip design. Moreover, whole-process generation can globally optimize the design and thus find the globally optimal layout. However, the whole-process end-to-end generation is much more difficult than both the single-stage and the cross-stage end-to-end generation because of the high complexity of design space, where the complexity is the product of the complexity of all stages. Therefore, the exploration in such a direction is greatly challenging.

(4) Practicability improvement. The practicability of chip design with ML can be improved by exploring two directions (i.e., to address Challenge-3). First, the chip design scale with ML must

be improved. In fact, the design scale of most existing studies is less than that of current real chips. Concretely, the largest design scale of the existing studies is about tens of millions of transistors [58]. In contrast, the scale of current real chips is about billions of transistors [1]. For example, the A100 GPU chip released by Nvidia in 2020 integrates more than 50 billion transistors. Therefore, improving the scale of chip design with ML can significantly enhance the practicability and is promising for industrial deployment. Second, datasets and benchmarks based on real data can be proposed for chip design with ML. In recent years, several circuit benchmarks have been proposed for different problems in chip design. For example, ISPD 2005 contest benchmark [142] has been used to explore the global placement problem [58]. However, transforming these benchmarks into datasets is hard because of the lack of specifications and design rules. Besides, some benchmarks use designs that are not extracted from real circuits or chips, and therefore, it is difficult to transfer the experimental results to actual work. Moreover, most benchmarks are not open-source, which greatly hinders the reproduction of classical and SOTA work in chip design with ML.

(5) Others. Beyond end-to-end generation and practicability, ML algorithms can be applied to explore more directions of a chip. On the one hand, ML algorithms can be leveraged to automatically generate chip performance specifications, such as timing specifications, area specifications, and power specifications (i.e., to address Challenge-2). As a result, research and experiments on different chips can be greatly boosted since there is no need for iterations between chip design and writing chip performance specifications. On the other hand, chips with new functions can be explored and invented by using ML algorithms. In this way, chip development and invention are significantly facilitated because lots of time and human resources can be saved in designing new functions of chips.

Acknowledgements This work was partially supported by National Natural Science Foundation of China (Grant Nos. 61925208, 62222214, 62102399, U22A2028, U19B2019), Beijing Academy of Artificial Intelligence (BAAI), CAS Project for Young Scientists in Basic Research (Grant No. YSBR-029), and Youth Innovation Promotion Association CAS and Xplore Prize.

References

1 Chen Y, Du Z, Guo Q, et al. From chip design to chip learning (in Chinese). Bull Chin Academy Sci, 2022, 37: 15–23
2 Huang G, Hu J, He Y, et al. Machine learning for electronic design automation: a survey. ACM Trans Des Autom Electron Syst, 2021, 26: 1–46
3 Lopera D S, Servadei L, Kiprit G N, et al. A comprehensive survey on electronic design automation and graph neural networks: theory and applications. ACM Trans Design Autom Electron Syst, 2023, 28: 1–27
4 Yan J, Lyu X, Cheng R, et al. Towards machine learning for placement and routing in chip design: a methodological overview. 2022. ArXiv:2202.13564
5 Cristescu M C. Machine learning techniques for improving the performance metrics of functional verification. Sci Technol, 2021, 24: 99–116
6 Tang H, Liu G, Chen X, et al. A survey on Steiner tree construction and global routing for VLSI design. IEEE Access, 2020, 8: 68593–68622
7 Xie Z, Pan J, Chang C C, et al. The dark side: security concerns in machine learning for EDA. 2022. ArXiv:2022.10597
8 Scarselli F, Gori M, Tsoi A C, et al. The graph neural network model. IEEE Trans Neural Netw, 2008, 20: 61–80
9 Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial networks. Commun ACM, 2020, 63: 139–144
10 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. 770–778
11 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, 2017. 6000–6010
12 Watkins C J C H, Dayan P. Q-learning. Machine learning. 1992, 8: 279–292
13 Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. 2013. ArXiv:1312.5602
14 Ferianc M, Fan H X, Chu R, et al. Improving performance estimation for FPGA-based accelerators for convolutional neural networks. In: Proceedings of the 16th International Symposium on Applied Reconfigurable Computing, Toledo, 2020. 3–13
15 Dai S, Zhou Y, Zhang H, et al. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In: Proceedings of the 26th IEEE International Symposium on Field-Programmable Custom Computing Machines, Boulder, 2018. 129–132
16 Zhao J, Liang T, Sinha S, et al. Machine learning based routing congestion prediction in FPGA high-level synthesis. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Florence, 2019. 1130–1135
17 Makrani H M, Sayadi H, Mohsenin T, et al. XPPE: cross-platform performance estimation of hardware accelerators using machine learning. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference, Tokyo, 2019. 727–732
18 Makrani H M, Farahmand F, Sayadi H, et al. Pyramid: machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design. In: Proceedings of the 29th International Conference on Field-Programmable Logic and Applications, Barcelona, 2019. 397–403
19 De S, Shafique M, Corporaal H. Delay prediction for ASIC HLS: comparing graph-based and nongraph-based learning models. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2023, 42: 1133–1146
20 Que Y H, Kapre N, Ng H, et al. Improving classification accuracy of a machine learning approach for FPGA timing closure. In: Proceedings of the IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines, Washington, 2016. 80–83
21 Liu H Y, Carloni L P. On learning-based methods for design-space exploration with high-level synthesis. In: Proceedings of the 50th Annual Design Automation Conference, New York, 2013. 1–7

22 Mahapatra A, Schafer B C. Machine-learning based simulated annealer method for high level synthesis design space exploration. In: Proceedings of the Electronic System Level Synthesis Conference, San Francisco, 2014. 1–6
23 Kwon J, Carloni L P. Transfer learning for design-space exploration with high-level synthesis. In: Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD, Virtual, 2020. 163–168
24 Wu N, Xie Y, Hao C. IronMan: GNN-assisted design space exploration in high-level synthesis via reinforcement learning. In: Proceedings of the 2021 on Great Lakes Symposium on VLSI, Virtual, 2021. 39–44
25 Cheng W K, Guo Y Y, Wu C S. Evaluation of routability-driven macro placement with machine-learning technique. In: Proceedings of the 7th International Symposium on Next-Generation Electronics, Taipei, 2018. 1–3
26 Huang Y H, Xie Z, Fang G Q, et al. Routability-driven macro placement with embedded CNN-based prediction model. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Florence, 2019. 180–185
27 Chan W T J, Chung K Y, Kahng A B, et al. Learning-based prediction of embedded memory timing failures during initial floorplan design. In: Proceedings of the 21st Asia and South Pacific Design Automation Conference, Macao, 2016. 178–185
28 Zhang S Z, Zhao Z Y, Feng C C, et al. A machine learning framework with feature selection for floorplan acceleration in ic physical design. J Comput Sci Technol, 2020, 35: 468–474
29 Shanthi J, Rani D G N, Rajaram S. A C4.5 decision tree classifier based floorplanning algorithm for system-on-chip design. MicroElectron J, 2022, 121: 105361
30 He Z, Ma Y, Zhang L, et al. Learn to floorplan through acquisition of effective local search heuristics. In: Proceedings of IEEE 38th International Conference on Computer Design, Hartford, 2020. 324–331
31 Oh C, Bondesan R, Kianfar D, et al. Bayesian optimization for macro placement. 2022. ArXiv:2207.08398
32 Liu Y, Ju Z, Li Z, et al. Floorplanning with graph attention. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, 2022. 1303–1308
33 Liu Y, Ju Z, Li Z, et al. GraphPlanner: floorplanning with graph neural network. ACM Trans Des Autom Electron Syst, 2023, 28: 1–24
34 Mirhoseini A, Goldie A, Yazgan M, et al. Chip placement with deep reinforcement learning. 2020. ArXiv:2004.10746
35 Xu Q, Geng H, Chen S, et al. GoodFloorplan: graph convolutional network and reinforcement learning-based floorplanning. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2022, 41: 3492–3502
36 Amini M, Zhang Z, Penmetsa S, et al. Generalizable floorplanner through corner block list representation and hypergraph embedding. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, 2022. 2692–2702
37 Chang F C, Tseng Y W, Yu Y W, et al. Flexible multiple-objective reinforcement learning for chip placement. 2022. ArXiv:2204.06407
38 Barboza E C, Shukla N, Chen Y, et al. Machine learning-based pre-routing timing prediction with reduced pessimism. In: Proceedings of the 56th Annual Design Automation Conference, Las Vegas, 2019. 1–6
39 Ho C T, Kahng A B. IncPIRD: fast learning-based prediction of incremental IR drop. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, Westminster, 2019. 1–8
40 Guo Z, Liu M, Gu J, et al. A timing engine inspired graph neural network model for pre-routing slack prediction. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, 2022. 1207–1212
41 Yang T, He G, Cao P. Pre-routing path delay estimation based on transformer and residual framework. In: Proceedings of the 27th Asia and South Pacific Design Automation Conference, Taipei, 2022. 184–189
42 Tabrizi A F, Darav N K, Xu S, et al. A machine learning framework to identify detailed routing short violations from a placed netlist. In: Proceedings of the 55th Annual Design Automation Conference, San Francisco, 2018. 1–6
43 Xie Z, Huang Y H, Fang G Q, et al. RouteNet: routability prediction for mixed-size designs using convolutional neural network. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, San Diego, 2018. 1–8
44 Liang R, Xiang H, Pandey D, et al. DRC hotspot prediction at sub-10nm process nodes using customized convolutional network. In: Proceedings of the International Symposium on Physical Design, Taipei, 2020. 135–142
45 Liu S, Sun Q, Liao P, et al. Global placement with deep learning-enabled explicit routability optimization. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Virtual, 2021. 1821–1824
46 Ghose A, Zhang V, Zhang Y, et al. Generalizable cross-graph embedding for GNN-based congestion prediction. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, Munich, 2021. 1–9
47 Wang B, Shen G, Li D, et al. LHNN: lattice hypergraph neural network for VLSI congestion prediction. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, 2022. 1–6
48 Xie Z, Ren H, Khailany B, et al. PowerNet: transferable dynamic IR drop estimation via maximum convolutional neural network. In: Proceedings of the 25th Asia and South Pacific Design Automation Conference, Beijing, 2020. 13–18
49 Zhou H, Jin W, Tan S X D. GridNet: fast data-driven EM-induced IR drop prediction and localized fixing for on-chip power grid networks. In: Proceedings of the 39th International Conference on Computer-Aided Design, Virtual, 2020. 1–9
50 Cao Y, Kahng A B, Li J, et al. Learning-based prediction of package power delivery network quality. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference, Tokyo, 2019. 160–166
51 Hyun D, Fan Y, Shin Y. Accurate wirelength prediction for placement-aware synthesis through machine learning. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Florence, 2019. 324–327
52 Lu Y C, Pentapati S, Lim S K. The law of attraction: affinity-aware placement optimization using graph neural networks. In: Proceedings of the International Symposium on Physical Design, Virtual, 2021. 7–14
53 Guo Z, Lin Y. Differentiable-timing-driven global placement. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, 2022. 1315–1320
54 Zhou X, Ye J, Pui C W, et al. Heterogeneous graph neural network-based imitation learning for gate sizing acceleration. In: Proceedings of International Conference on Computer-Aided Design, San Diego, 2022. 1–9
55 Vashisht D, Rampal H, Liao H, et al. Placement in integrated circuits using cyclic reinforcement learning and simulated annealing. 2020. ArXiv:2011.07577
56 Agnesina A, Chang K, Lim S K. VLSI placement parameter optimization using deep reinforcement learning. In: Proceedings of the 39th International Conference on Computer-Aided Design, Virtual, 2020. 1–9
57 Lu Y C, Nath S, Khandelwal V, et al. RL-Sizer: VLSI gate sizing for timing optimization using deep reinforcement learning. In: Proceedings of the 58th ACM/IEEE Design Automation Conference, San Francisco, 2021. 733–738
58 Lin Y, Dhar S, Li W, et al. DREAMPlace: deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In: Proceedings of the 56th Annual Design Automation Conference, Las Vegas, 2019. 1–6
59 Liu L, Fu B, Wong M D F, et al. Xplace: an extremely fast and extensible global placement framework. In: Proceedings of

the 59th ACM/IEEE Design Automation Conference, San Francisco, 2022. 1309–1314
60 Nath S, Pradipta G, Hu C, et al. Generative self-supervised learning for gate sizing. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, 2022. 1331–1334
61 Chhabria V A, Kahng A B, Kim M, et al. Template-based PDN synthesis in floorplan and placement using classifier and CNN techniques. In: Proceedings of the 25th Asia and South Pacific Design Automation Conference, Beijing, 2020. 44–49
62 Qi Z, Cai Y, Zhou Q. Accurate prediction of detailed routing congestion using supervised data learning. In: Proceedings of IEEE 32nd International Conference on Computer Design, Seoul, 2014. 97–103
63 Zhou Q, Wang X, Qi Z, et al. An accurate detailed routing routability prediction model in placement. In: Proceedings of the 6th Asia Symposium on Quality Electronic Design, Kula Lumpur, 2015. 119–122
64 He Y, Bao F S. Circuit routing using Monte Carlo tree search and deep neural networks. 2020. ArXiv:2006.13607
65 Chen J, Kuang J, Zhao G, et al. PROS: a plug-in for routability optimization applied in the state-of-the-art commercial EDA tool using deep learning. In: Proceedings of the 39th International Conference on Computer-Aided Design, Virtual, 2020. 1–8
66 Liao H, Dong Q, Dong X, et al. Attention routing: track-assignment detailed routing using attention-based reinforcement learning. In: Proceedings of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, St. Louis, 2020. 84003: V11AT11A002
67 Ju X, Zhu K, Lin Y, et al. Asynchronous multi-nets detailed routing in VLSI using multi-agent reinforcement learning. In: Proceedings of the 7th IEEE International Conference on Network Intelligence and Digital Content, Beijing, China, 2021. 250–254
68 Lin Y, Qu T, Lu Z, et al. Asynchronous reinforcement learning framework and knowledge transfer for net-order exploration in detailed routing. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2022, 41: 3132–3142
69 Utyamishev D, Partin-Vaisband I. Late breaking results: a neural network that routes ICs. In: Proceedings of the 57th ACM/IEEE Design Automation Conference, Virtual, 2020. 1–2
70 Hosny A, Hashemi S, Shalan M, et al. DRiLLS: deep reinforcement learning for logic synthesis. In: Proceedings of the 25th Asia and South Pacific Design Automation Conference, Beijing, 2020. 581–586
71 Grosnit A, Malherbe C, Tutunov R, et al. BOiLS: Bayesian optimisation for logic synthesis. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Virtual, 2022. 1193–1196
72 Floridi L, Chiriatti M. GPT-3: its nature, scope, limits, and consequences. Minds Machines, 2020, 30: 681–694
73 Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529: 484–489
74 Jumper J, Evans R, Pritzel A, et al. Highly accurate protein structure prediction with AlphaFold. Nature, 2021, 596: 583–589
75 Fawzi A, Balog M, Huang A, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 2022, 610: 47–53
76 Ambasana N, Gope D, Mutnury B, et al. Application of artificial neural networks for eye-height/width prediction from S-parameters. In: Proceedings of IEEE 23rd Conference on Electrical Performance of Electronic Packaging and Systems, 2014. 99–102
77 Lu T, Wu K, Yang Z, et al. High-speed channel modeling with deep neural network for signal integrity analysis. In: Proceedings of the 26th Conference on Electrical Performance of Electronic Packaging and Systems, San Jose, 2017. 1–3
78 Kwon Y, Jung J, Han I, et al. Transient clock power estimation of pre-CTS netlist. In: Proceedings of IEEE International Symposium on Circuits and Systems, Florence, 2018. 1–4
79 Lee Y Y, Ruan S J, Chen P C. Predictable coupling effect model for global placement using generative adversarial networks with an ordinary differential equation solver. IEEE Trans Circuits Syst II, 2021, 69: 2261–2265
80 Xie Z, Liang R, Xu X, et al. Net^2: a graph attention network method customized for pre-placement net length estimation. In: Proceedings of the 26th Asia and South Pacific Design Automation Conference, Taipei, 2021. 671–677
81 Neto W L, Austin M, Temple S, et al. LSOracle: a logic synthesis framework driven by artificial intelligence. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, Westminster, 2019. 1–6
82 Ma Y, Ren H, Khailany B, et al. High performance graph convolutional networks with applications in testability analysis. In: Proceedings of the 56th Annual Design Automation Conference, Las Vegas, 2019. 1–6
83 Gad M, Aboelmaged M, Mashaly M, et al. Efficient sequence generation for hardware verification using machine learning. In: Proceedings of the 28th IEEE International Conference on Electronics, Circuits, and Systems, Dubai, 2021. 1–5
84 Haaswijk W, Collins E, Seguin B, et al. Deep learning for logic optimization algorithms. In: Proceedings of IEEE International Symposium on Circuits and Systems, Florence, 2018. 1–4
85 Zhu K, Liu M, Chen H, et al. Exploring logic optimizations with reinforcement learning and graph convolutional network. In: Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD, Virtual, Iceland, 2020. 145–150
86 Shi Z, Li M, Khan S, et al. DeepTPI: test point insertion with deep reinforcement learning. 2022. ArXiv:2206.06975
87 Balog M, Gaunt A L, Brockschmidt M, et al. DeepCoder: learning to write programs. In: Proceedings of the 5th International Conference on Learning Representations, Toulon, 2017. 1–21
88 Lu Y C, Lee J, Agnesina A, et al. GAN-CTS: a generative adversarial framework for clock tree prediction and optimization. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, Westminster, 2019. 1–8
89 Koh S, Kwon Y, Shin Y. Pre-layout clock tree estimation and optimization using artificial neural network. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, Boston, 2020. 193–198
90 Zheng X, Eder K, Blackmore T. A neural network based novel test selector. 2022. ArXiv:2207.00445
91 Donno M, Ivaldi A, Benini L, et al. Clock-tree power optimization based on RTL clock-gating. In: Proceedings of the 40th annual Design Automation Conference, Anaheim, 2003. 622–627
92 Chan H G, Goh P. Neural networks for eye height and eye width prediction with an improved adaptive sampling algorithm. In: Proceedings of the 17th Asian Simulation Conference, Melaka, 2017. 189–201
93 Enzler R, Jeger T, Cottet D, et al. High-level area and performance estimation of hardware building blocks on FPGAs. In: Proceedings of Field-Programmable Logic and Applications, Villach, 2000. 525–534
94 Qi Z, Cai Y, Zhou Q, et al. VFGR: a very fast parallel global router with accurate congestion modeling. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference, Tokyo, 2014. 525–530
95 Chang H, Sapatnekar S S. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, 2004. 621–625

96 Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, 2016. 785–794
97 Salvatori T, Song Y, Xu Z, et al. Reverse differentiation via predictive coding. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence, Virtual, 2022. 8150–8158
98 Ororbia A, Kifer D. The neural coding framework for learning generative models. Nat Commun, 2022, 13: 2064
99 Hinton G. The forward-forward algorithm: some preliminary investigations. 2022. ArXiv:2212.13345
100 Ororbia A, Mali A. The predictive forward-forward algorithm. 2023. ArXiv:2301.01452
101 Spindler P, Johannes F M. Fast and accurate routing demand estimation for efficient routability-driven placement. In: Proceedings of the Conference on Design, Automation and Test in Europe, Nice, 2007. 1–6
102 Ronneberger O, Fischer P, Brox T. U-Net: convolutional networks for biomedical image segmentation. In: Proceedings of the 18th International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, 2015. 234–241
103 Huang C C, Lee H Y, Lin B Q, et al. NTUplace4dr: a detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2017, 37: 669–681
104 Grover A, Leskovec J. Node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, 2016. 855–864
105 Tang J, Qu M, Wang M, et al. LINE: large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, Florence, 2015. 1067–1077
106 Perozzi B, Al-Rfou R, Skiena S. DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, 2014. 701–710
107 Isola P, Zhu J Y, Zhou T, et al. Image-to-image translation with conditional adversarial networks. In: Proceedings of Conference on Computer Vision and Pattern Recognition, Honolulu, 1125-1134
108 Fang Y C, Lin H Y, Sui M Y, et al. Machine-learning-based dynamic IR drop prediction for ECO. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Diego, 2018. 1–7
109 Sun Z, Yu S, Zhou H, et al. EMSpice: physics-based electromigration check using coupled electronic and stress simulation. IEEE Trans Device Mater Relib, 2020, 20: 376–389
110 Dai K R, Liu W H, Li Y L. NCTU-GR: efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing. IEEE Trans VLSI Syst, 2011, 20: 459–472
111 Hu B, Marek-Sadowska M. Wire length prediction based clustering and its application in placement. In: Proceedings of the 40th annual Design Automation Conference, Anaheim, 2003. 800–805
112 Kahng A B, Reda S. Intrinsic shortest path length: a new, accurate a priori wirelength estimator. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, 2005. 173–180
113 Fathi B, Behjat L, Rakai L M. A pre-placement net length estimation technique for mixed-size circuits. In: Proceedings of the 11th International Workshop on System Level Interconnect Prediction, Francisco, 2009. 45–52
114 Freund Y, Schapire R, Abe N. A short introduction to boosting. J-Japan Soc Artif Intell, 1999, 14: 1612
115 Hastie T, Tibshirani R, Friedman J H, et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, 2009. 1–698
116 Krishnamurthy B. A dynamic programming approach to the test point insertion problem. In: Proceedings of the 24th ACM/IEEE Design Automation Conference, Miami Beach, 1987. 695–705
117 Yu K, Bi J, Tresp V. Active learning via transductive experimental design. In: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, 2006. 1081–1088
118 Quinlan J. C4.5: Programs for Machine Learning. San Francisco: Morgan Kaufmann Publishers, 1993. 155–164
119 Flach G, Reimann T, Posser G, et al. Effective method for simultaneous gate sizing and V th assignment using lagrangian relaxation. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2014, 33: 546–557
120 Browne C B, Powley E, Whitehouse D, et al. A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games, 2012, 4: 1–43
121 Yang W, Wang L, Mishchenko A. Lazy man's logic synthesis. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, 2012. 597–604
122 Amarú L, Gaillardon P E, de Micheli G. The EPFL combinational benchmark suite. In: Proceedings of the 24th International Workshop on Logic & Synthesis, Mountain View, 2015. 1–5
123 Niu S, Chen S, Guo H, et al. Generalized value iteration networks: life beyond lattices. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, New Orleans, 2018. 6246–6253
124 Sykora Q, Ren M, Urtasun R. Multi-agent routing value iteration network. In: Proceedings of the 37th International Conference on Machine Learning, Virtual, 2020. 9300–9310
125 Qu T, Lin Y, Lu Z, et al. Asynchronous reinforcement learning framework for net order exploration in detailed routing. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Virtual, 2021. 1815–1820
126 Ansel J, Kamil S, Veeramachaneni K, et al. OpenTuner: an extensible framework for program autotuning. In: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, Edmonton, 2014. 303–316
127 Schotten C, Meyr H. Test point insertion for an area efficient BIS. In: Proceedings of the IEEE International Test Conference on Driving Down the Cost of Test, Washington, 1995. 515–523
128 Rai S, Neto W L, Miyasaka Y, et al. Logic synthesis meets machine learning: trading exactness for generalization. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Virtual, 2021. 1026–1031
129 Miyasaka Y, Zhang X, Yu M, et al. Logic synthesis for generalization and learning addition. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Virtual, 2021. 1032–1037
130 Cowen-Rivers A I, Lyu W, Tutunov R, et al. An empirical study of assumptions in Bayesian optimisation. 2020. ArXiv:2012.03826
131 Oh C, Bondesan R, Gavves E, et al. Batch Bayesian optimization on permutations using acquisition weighted kernels. 2021. ArXiv:2102.13382
132 Veličković P, Cucurull G, Casanova A, et al. Graph attention networks. In: Proceedings of the 6th International Conference on Learning Representations, Vancouver, 2018. 1–12
133 Gu J, Jiang Z, Lin Y, et al. DREAMPlace 3.0: multi-electrostatics based robust VLSI placement with region constraints. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, Virtual, 2020. 1–9
134 Lu J, Chen P, Chang C C, et al. ePlace: electrostatics-based placement using fast Fourier transform and Nesterov's method. ACM Trans Des Autom Electron Syst, 2015, 20: 1–34
135 Cheng C K, Kahng A B, Kang I, et al. RePlAce: advancing solution quality and routability validation in global placement.

IEEE Trans Comput-Aided Des Integr Circuits Syst, 2018, 38: 1717–1730
136 Lin Y, Pan D Z, Ren H, et al. DREAMPlace 2.0: open-source GPU-accelerated global and detailed placement for large-scale VLSI designs. In: Proceedings of China Semiconductor Technology International Conference, Shanghai, 2020. 1–4
137 Liao P, Liu S, Chen Z, et al. DREAMPlace 4.0: timing-driven global placement with momentum-based net weighting. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Virtual, 2022. 939–944
138 Xu Y, Zhang Y, Chu C. FastRoute 4.0: global router with efficient via minimization. In: Proceedings of the Asia and South Pacific Design Automation Conference, Yokohama, 2009. 576–581
139 Chen T-C, Chang Y-W. Modern floorplanning based on B*-tree and fast simulated annealing. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2006, 25: 637–650
140 Xu Q, Chen S, Li B. Combining the ant system algorithm and simulated annealing for 3D/2D fixed-outline floorplanning. Appl Soft Computing, 2016, 40: 150–160
141 Cheng R, Yan J. On joint learning for solving placement and routing in chip design. In: Proceedings of Advances in Neural Information Processing Systems, 2021. 34: 16508–16519
142 Nam G J, Alpert C J, Villarrubia P, et al. The ISPD2005 placement contest and benchmark suite. In: Proceedings of the International Symposium on Physical Design, San Francisco, 2005. 216–220