A Survey of Machine Learning Approaches in Logic Synthesis

MIAO LIU, University of Chinese Academy of Sciences, China
LIWEI NI, Institute of Computing Technology, Chinese Academy of Sciences, China
JUNFENG LIU, Pengcheng Laboratory, China
XINGYU MENG, Pengcheng Laboratory, China
RUI WANG, Shenzhen University, China
XIAOZE LIN, Institute of Computing Technology, Chinese Academy of Sciences, China
XINHUA LAI, University of Chinese Academy of Sciences, China
XINGQUAN LI*, Pengcheng Laboratory, China
JUNGANG XU*, University of Chinese Academy of Sciences, China

The increasing complexity of digital circuits and the limitations of heuristic methods have led to growing interest in applying Machine Learning (ML) to Logic Synthesis (LS). ML provides a promising paradigm shift by implementing automated, scalable, and data-driven optimization strategies. This survey provides a comprehensive overview of the latest studies on ML approaches in LS, offering a deep understanding of the fundamental ML methods, and analyzing their strengths and limitations through systematic comparisons. We categorize existing works into two main types: ML-Assistance methods aiming at predicting performance metrics and reducing the cost of traditional simulations, and ML-Agent methods directly replacing heuristic processes in the LS flow. We further analyze ML methods and applications in different LS stages, including Boolean circuit generation, Boolean circuit analysis, logic optimization, and technology mapping, showing great achievements and improvement in exploring non-linear design spaces and discovering new optimization strategies. Finally, we discuss the challenges and limitations in the current situation and further provide a vision of future directions in LS.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Hardware → Logic synthesis**; • **Computing methodologies → Machine learning**.

ACM Reference Format:
Miao Liu, Liwei Ni, Junfeng Liu, Xingyu Meng, Rui Wang, Xiaoze Lin, Xinhua Lai, Xingquan Li, and Jungang Xu. 2025. A Survey of Machine Learning Approaches in Logic Synthesis. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1, Article 1 (January 2025), 41 pages. https://doi.org/XXXXXXX.XXXXXXX

*Corresponding author

This work is supported in part by the Major Key Project of PCL (No. PCL2025AS04) and the NSF of Fujian Province under Grants (No. 2024J09045).
Authors' Contact Information: Miao Liu, liumiao20@mails.ucas.ac.cn, University of Chinese Academy of Sciences, Beijing, China; Liwei Ni, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; Junfeng Liu, Pengcheng Laboratory, Shenzhen, Guangdong sheng, China; Xingyu Meng, Pengcheng Laboratory, Shenzhen, Guangdong sheng, China; Rui Wang, Shenzhen University, Shenzhen, Guangdong sheng, China; Xiaoze Lin, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; Xinhua Lai, University of Chinese Academy of Sciences, Beijing, China; Xingquan Li, Pengcheng Laboratory, Shenzhen, Guangdong sheng, China, fzulxq@gmail.com; Jungang Xu, University of Chinese Academy of Sciences, Beijing, China, xujg@ucas.ac.cn.

ACM 1557-7309/2025/1-ART1
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In recent years, Machine Learning (ML) has made significant strides in various fields, and its applications in Electronic Design Automation (EDA) have been no exception [127]. One of the most promising areas where ML has begun to revolutionize the industry is in Logic Synthesis (LS). LS, the process of transforming high-level descriptions of digital systems into gate-level implementations, has traditionally been a highly complex and time-consuming task [60, 65, 104]. This complexity arises from not only the need to simultaneously optimize multiple conflicting objectives adhering to various constraints, such as Performance, Power, and Area (PPA), but also the difficulty of cross-stage evaluation, where the impact of decisions made in earlier stages cannot be accurately assessed in later stages of the synthesis flow.

Conventionally, Logic Optimization (LO) relies on heuristics, exact transformation and intuitions by human experts [65, 107], which requires expertise in both LS and corresponding commercial tools. Meanwhile, these traditional heuristic-based designs cannot guarantee optimality, especially in the case of increasingly complicated Boolean functions. As such, it seems natural to move towards more automatic and powerful methodologies for LS, and the relationship between ML and LS is being reconsidered. There has been a recent emergence of applying ML in LS, which embraces a twofold meaning: (1) the reduction of burdens on human experts designing optimization recipes manually, and (2) enabling the discovery of novel optimization strategies that extend beyond conventional human intuitions [127, 180]. By leveraging ML algorithms, the field has the potential to revolutionize LS. Such integration is expected to significantly reduce the design time and cost, while pushing the boundaries of what is achievable in LO.

Existing works applying ML to LS vary based on the application tasks. To provide a clear taxonomy, we categorize them into two main types based on the role and goal of the ML model in the synthesis flow:

- **ML-Assistance**: In this paradigm, the ML model acts as a helper tool to assist traditional algorithms. It typically predicts intermediate metrics (e.g., Quality of Result (QoR), Satisfiability (SAT), or cut quality) to guide a traditional, deterministic engine, such as the synthesis tool [4, 164]. The traditional algorithm retains control over the final decision-making but operates more efficiently within a refined search space provided by the ML model.
- **ML-Agent**: In this paradigm, the ML model acts as a decision maker or executor that replaces the traditional heuristic process. It directly completes the task, such as generating a circuit structure [126], exploring optimization recipes [56], or predicting the final output directly, without relying on a traditional algorithm to execute the core logic.

While other excellent reviews like [16] exist, this survey provides several key additional contributions: (1) We strictly structure our analysis according to the LS flow, as shown in Fig. 1, dedicating discussion to each stage. (2) This structure allows us to incorporate a dedicated section on "Boolean Circuit Analysis", covering the circuit representation learning, SAT, Design for Testability (DFT), and Security, which was not detailed in [16] but is critical to the modern LS flow. (3) We also provide a detailed review of the emerging field of "Logic Learning", with a specific focus on recent progress in generating functionally exact circuits from complete specifications, distinguishing our work from [16] that focused on approximate synthesis from incomplete specifications.

This article provides a comprehensive overview of the latest advancements in ML-based methods and applications for LS. We will explore how ML models are integrated into various stages of the synthesis process, such as LO, Technology Mapping (TM), and other tasks. Fig. 1 briefly shows the LS flow and the data format flow, according to which we organize this article. Table 1 presents a general summary of the ML tasks, techniques, and applications reviewed in this article, categorized by their corresponding LS stage. In Section 3, we first consider Boolean circuit generation, which,

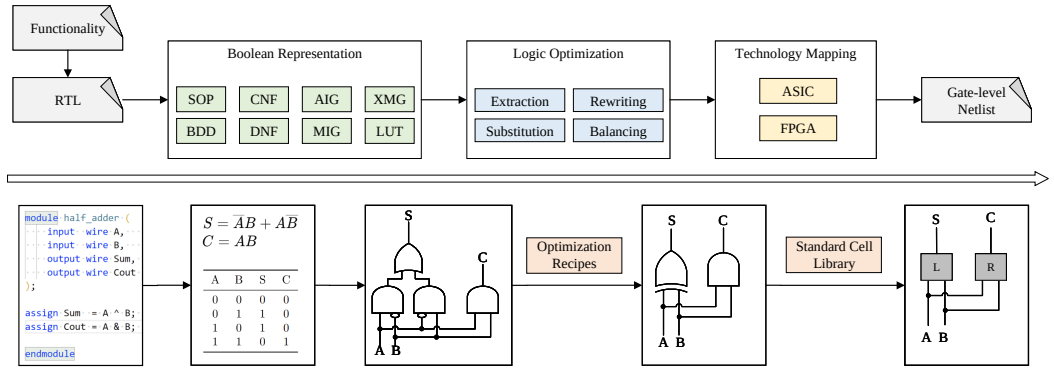

Fig. 1. Overview of the logic synthesis flow and data format flow.

although not part of the typical flow, plays a significant role in LS. We survey some recent works on the logic learning problem. In Section 4, we consider the ML applications in the tasks of Boolean circuit representation, which mainly analyze the characteristics of the circuits. We introduce the problems and ML methods of circuit representation learning, which can be applied further in extension problems, such as SAT, DFT, and security. We also survey some representative works in ML methods addressing these problems. In Section 5, we consider the ML tasks in LO, including the QoR prediction and recipe exploration task that is ML-Agent to directly help optimization, and the Boolean classification task, which is ML-Assistance and indirectly applied in optimization. In Section 6, we also provide a general discussion on ML applications in TM, which are mainly used for cut selection. Finally, we review the studies associated with ML tools and ML dataset management. Finally, we discuss the limitations of existing works and future prospects about ML applications in LS.

2 Preliminaries

2.1 Logic Synthesis

LS is a foundational and critical technology in EDA that transforms a Register Transfer Level (RTL) description of a circuit into gate-level representations of a Boolean circuit, then into a standard cell-level netlist that is ready for physical design and manufacturing. The LS process plays an important role in ensuring that the final netlist not only meets functional correctness, but also adheres to design constraints. This process generally involves three main stages: defining the circuit's Boolean representation, performing logic optimization, and executing technology mapping.

2.1.1 Boolean Representation. The first step in LS involves modeling the logical functionality of a digital circuit in a formal and manageable format [60]. The Boolean representation is the abstract foundation where LO is performed, including:

- Sum-of-Products (SOP) : A traditional expression using logical AND and OR operations, such as $S = AB + \overline{A}C$. SOPs are intuitive but often inefficient for large circuits due to a lack of structure and redundancy. A related form is Conjunctive Normal Form (CNF).
- Binary Decision Diagram (BDD) : A graphical representation of Boolean functions that applies decision trees and supports equivalence checking and formal verification.
- Boolean Network : A graphical representation that uses Directed Acyclic Graphs (DAGs), is suitable for large-scale designs. Each node represents a logic operation (e.g., AND, OR) or

a Primary Input/Output (PI/PO). Each edge represents a signal or an inverted connection between operations.
- Look-Up Table (LUT) : Each LUT implements a small Boolean function by storing output values for all possible input combinations. The size of a LUT is typically fixed (e.g., 4-input or 6-input), and larger functions are constructed by combining multiple LUTs.

An important Boolean network is the And-Inverter Graph (AIG), which consists solely of two-input AND gates and inverters (logical NOT) [107]. AIGs offer a compact and uniform representation that is convenient for LO, making them popular in recent works. Fig. 2 provides an illustration of an AIG representation and the result of its optimization. An alternative representation that is designed for emerging technologies, such as spin-wave devices and quantum-dot cellular automata, is the Majority-Inverter Graph (MIG)[10, 85]. MIG uses majority logic and inverters, and is also a compact and uniform structure. Furthermore, some representations can extend AIGs and MIGs by incorporating XOR gates, resulting in XOR-And Graphs (XAGs) and XOR-Majority Graphs (XMGs) [35].

2.1.2 Logic Optimization. Once the Boolean representation is established, the next stage focuses on LO. The goal is to transform the initial, suboptimal logic circuit into a more efficient form without changing its functional behavior. LO aims to improve several circuit characteristics, including: reducing logic depth, minimizing the number of logic gates and switching activity. Since these objectives are tightly coupled with the PPA metrics, LO plays a central role in high-quality synthesis.

LO is typically performed on the technology-independent Boolean networks. Many DAG-aware algorithms have been developed, particularly targeting AIGs, including:

- Rewrite (rw): Rewriting performs template-based pattern matching on subgraphs of the AIG, replacing them with functionally equivalent but potentially more optimized structures, which are called Negation-Permutation-Negation (NPN) equivalence classes. The rewriting process involves k-feasible cut enumeration, where each cut is analyzed to identify subgraphs to be replaced [107].
- Refactor (rf): Refactoring is the extension of the rewrite, which targets large cuts or Maximum Fanout-Free Cones (MFFCs), collapses them into an SOP or truth table, and then applies minimization and factoring to dynamically create a new, optimized logical structure.
- Balance (b): Balancing aims to minimize the logical depth (and thus delay) by restructuring the tree topology of the AIG. The process involves applying covering and tree-balancing transformations that leverage the associative and commutative properties of logic functions [106].
- Re-substitution (rs): Re-substitution aims to reduce redundancy and enhance logic sharing by re-expressing a target node function by reusing other already existing nodes in the network to discover and exploit global logic sharing [18].

These algorithms are combined sequentially to form *synthesis recipes* to optimize the circuit in LO. The effectiveness of a recipe depends on how well it balances conflicting optimization goals. For example, reducing depth may increase gate count, so the recipes are crucial to achieving the best trade-off.

2.1.3 Technology Mapping. The final stage is TM, which converts the optimized logic representation into a technology-specific netlist composed of standard cells from a technology library [154]. This step binds the technology-independent circuit to a real-world netlist implementation. During this phase, the circuit is: (1) partitioned based on cut enumeration, and each of the partitions performs a specific logic function, and then (2) matched to a standard cell, which includes both the primitive gates (e.g., AND, OR, XOR) and complex components (e.g., adders, multiplexers). Each match must

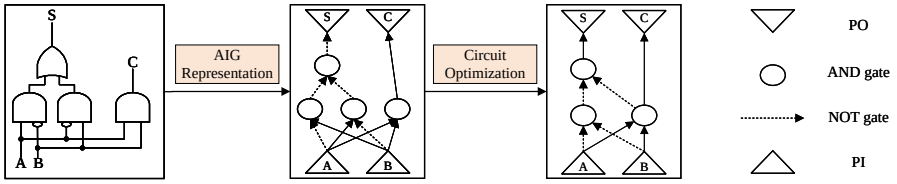

Fig. 2. AIG representation and its optimization result.

satisfy the PPA constraints imposed by the design specification. For instance, a subgraph with a critical timing path may need to be matched with a high-speed gate, while a non-critical path can be mapped to a low-power cell.

The goal of TM is to generate a fabrication-ready netlist that maintains the optimized logic structure. If the netlist fails to satisfy the PPA constraints, additional post-mapping optimizations are triggered to adjust the design. Otherwise, the RTL description must be revised, or the LO stage must be rerun to return to the synthesis flow. Then netlist becomes the input to downstream stages such as placement, routing, and physical verification.

2.2 Different ML Techniques

ML can be categorized into three frameworks: Supervised Learning, Unsupervised Learning, and Reinforcement Learning (RL). These frameworks mainly differ in what data are sampled and how these data are used to build learning models. In some cases, multiple models may be effective for a given problem, and the optimal choice depends on factors such as available hardware resources, data availability, implementation overhead, and performance targets.

2.2.1 Supervised Learning. Supervised learning involves learning rules from labeled datasets that map inputs to outputs, enabling the model to generalize and make predictions for unseen inputs. We briefly introduce several prevalent techniques in supervised learning.

- Regression is used to estimate the relationships between a dependent variable and one or more independent variables. The most common form is linear regression [136], alongside various non-linear regression methods [129].
- Support Vector Machine (SVM) [135] aims to identify the best hyperplanes to separate data classes by maximizing margins. Predictions or classifications of new inputs can be decided by their relative positions to these hyperplanes.
- Decision tree is used to build a tree structure where each node represents a feature and branches correspond to feature values, guiding inputs from the root to a leaf that delivers a prediction.
- Neural Networks (NNs) [58] are used to approximate a broad family of functions: a single-layer perceptron is usually used for linear tasks. Deep Neural Networks (DNNs) [54], consisting of multiple layers, are able to capture non-linear relations. Certain computational operations can help DNNs to achieve better performance in specific fields: Convolutional Neural Networks (CNNs) with convolution operations leveraging spatial features, Recurrent Neural Networks (RNNs) with recurrent connections enabling learning from sequences and histories, and Graph Neural Networks (GNNs) with aggregating and combining operations specifically on graphs.

2.2.2 Unsupervised Learning. Unsupervised learning is the process of identifying hidden patterns within unlabeled datasets. Two primary methods are clustering and Principal Component Analysis (PCA).

- Clustering is to group data objects into disjoint clusters based on similarity measurement, ensuring that objects within the same cluster exhibit high similarity, while those in different clusters have low similarity. The goal of clustering is to classify raw data reasonably and to uncover latent structures or patterns in datasets.
- PCA aims to reduce the dimensionality of the high-dimensional variable space by representing it with a few orthogonal variables that capture most of its variability. This dimensionality reduction simplifies data analysis while retaining essential information.

Self-supervised learning is a type of unsupervised learning that aims to learn a universal feature representation for downstream tasks by generating supervisory signals from the data itself. Self-supervised learning can be mainly divided into two categories: generative methods and contrastive methods.

- Generative learning focuses on understanding the underlying data distribution to generate new samples based on this understanding. Typically, the generative model predicts the probabilities of the next possible element in a sequence till completing the whole sequence.
- Contrastive Learning (CL) learns the effective representations of data by comparing the similarity between samples, without manual annotations. During training, the model generates multiple views of the same data (positive pairs) and treats different data views as negative pairs. By maximizing similarity between positive pairs and minimizing it between negative pairs, the model acquires discriminative feature representations.

2.2.3 Reinforcement Learning. In standard RL, an agent interacts with an environment $\mathcal{E}$ over a number of discrete time steps [152]. At each time step t, the agent receives a state s_t from the state space $\mathcal{S}$ and selects an action a_t from the action space $\mathcal{A}$ according to its policy π, where π is a mapping from states s_t to actions a_t. In return, the agent receives the next state s_{t+1} and a scalar reward $r_t : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. This process, which defines a single episode, continues until the agent reaches a terminal state, at which point the environment is reset for a new episode. The return R_t is the total accumulated discounted reward over an episode. The goal of the agent is to maximize the expected return for each state s. There are two general types of methods in RL: value-based and policy-based.

- In value-based RL, the state-action value function $Q_\pi(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$ is approximated by either tabular approaches or function approximations. At each state s_t , the agent always selects the optimal action a_t^* that could bring the maximal state-action value $Q(s_t, a_t^*)$.
- In policy-based RL, it directly parameterizes the policy $\pi(a|s; \theta)$ and updates the parameters θ by performing gradient ascent on $E[R_t]$.

RL is modeled based on the principles of Markov Decision Process (MDP), making it particularly well-suited for addressing control challenges and sequential decision-making tasks. Moreover, this alignment also allows RL to explore the design spaces by deciding the parameters sequentially. By embedding optimization objectives into reward functions, RL enables the discovery of optimal behaviors with specified goals.

3 Boolean Circuit Generation

Boolean circuit generation aims to construct a logic circuit that functionally implements a given specification, typically a truth table. Traditionally, this problem has been framed as "Exact Synthesis", which seeks to find a circuit with a guaranteed minimum of specific resources, such as area or

Table 1. General Summary of ML Tasks and Techniques Applied in LS.

Flow	Task	ML Domain	Technique	Input	Output	Type
Boolean Circuit Generation (section 3)	Logic Learning	AutoML	NAS, DNAS	Truth Table	Circuit	ML-Agent
		Graph Generative Learning	RL, MCTS, AR, GNN, Generative Transformer			
Boolean Circuit Analysis (section 4)	Circuit Representation Learning	Graph Representation Learning	GNN, GraphSAGE MPNN, DAG-GNN	Circuit	Embedding	ML-Agent
	Algorithm Engineering DFT, SAT, Security	Meta-Model, Meta-Learning	Task-Specific	Circuit	Task-Specific	Both
Logic Optimization (section 5)	QoR Prediction	Regression	Circuit: GNN Recipe: RNN, Transformer	Circuit Recipe	QoR	ML-Agent
	Boolean Representation Classification	Classification	CNN, GNN, Clustering	Subcircuits	Desired Representation	ML-Assistance
	Recipe Exploration	RL	RL, GNN	Circuit	Recipe	ML-Agent
		DSE	Bayesian Optimization, MCTS, NN			
	Algorithm Engineering Rewriting, Balancing	Meta-Model	RL	Circuit	Optimized Circuit	ML-Assistance
Technology Mapping (section 6)	Cut Selection	Classification	GNN, Transformer	Circuit	Priority Cuts	ML-Assistance
	Library Tuning	RL	RL	Library	Partial Library	ML-Agent
	Cost Function Modification	Regression	GNN	Cuts	Cost	ML-Assistance

delay. A truth table offers a complete and unambiguous representation of the function's behavior, serving as the ground truth for these generation tasks.

However, achieving a guaranteed optimal circuit is computationally intractable, with worst-case scenarios requiring exponential time. This complexity defines a long-standing tension between the search for optimality and the need for scalability. Early manual methods like Karnaugh maps (K-maps) provide visual minimization but become practically infeasible for functions beyond six variables [78]. Conversely, algorithms like Espresso offer efficient heuristics for two-level logic minimization but do not guarantee global optimality for general multi-level circuits. This dichotomy has motivated the search for new methodologies.

State-of-the-art exact synthesis often relies on SAT-based techniques, which transform the problem into a Boolean satisfiability instance and employ an iterative "trial-and-error" search to prove minimality. While capable of producing optimal results, the necessity of proving "UNSAT" for all smaller resource counts makes this approach computationally expensive and slow for large Boolean functions [147].

To address this bottleneck, ML techniques are being introduced as a powerful new paradigm for Boolean circuit generation. While ML-based methods do not strictly offer the mathematical proof of minimality inherent to exact synthesis, their fundamental objective is identical: to generate a functionally correct circuit that implements the target truth table. Often framed as "Logic Learning", these ML-Agent approaches leverage generative models to explore the design space efficiently, aiming to construct circuits that approximate optimal structures with significantly improved inference speed compared to traditional iterative solvers.

3.1 Logic Learning

The task of logic learning aims at transforming the design functionality into logic representations. This task was first proposed by the International Workshop on Logic & Synthesis (IWLS) 2020 contest [126] and subsequently matured in the IWLS 2022 contest [2]. In the IWLS 2020 contest [126], the provided dataset consisted of incomplete truth tables, where the goal was to learn an incomplete function. This task is closely related to approximate analysis, which trades exactness for generalization to achieve acceptable accuracy with reduced complexity. In contrast, the IWLS 2022 contest [2] provided complete truth tables, shifting the goal to learning the complete logic representation correspondingly. In this section, we discuss how recent ML techniques are applied to generate complete logic functions. Conversely, logic learning methods can also motivate advancements in ML theory, a topic surveyed in [16] and not elaborated further in this article.

Learning Boolean functions has long been an active area in theoretical ML, primarily under the Probably Approximately Correct (PAC) [67] and Statistical Query (SQ) learning frameworks [128]. Moreover, neural network-based methods are emerging as a significant direction in next-generation logic synthesis [162].

While truth tables fully specify the behavior of a Boolean function, they lack the structural details needed to derive the corresponding AIG directly. The challenge in logic learning is to synthesize the structure of the AIG that maps to the given truth table. More specifically, the core difficulty lies in enabling the ML model to recognize gate functionality and establish wire connections. Traditional ML models are inherently designed for continuous function approximation (learning input-output mappings) rather than constructing discrete DAGs. Due to this fundamental mismatch, ML architectures must be significantly adapted or used in entirely different ways to handle the discrete nature of logic gates and wires. To address this, two general strategies have emerged for circuit generation: sequential generation and architecture search, as illustrated in Fig. 3.

Problem Statement. Given a target truth table matrix $TT \in \{0,1\}^{2^n \times m}$, the objective is to construct an AIG with n inputs and m outputs, such that the truth table TT_o of the generated AIG satisfies either: (1) $TT_o = TT$ for exact generation (functional equivalence) or (2) TT_o partially matches TT for approximate generation.

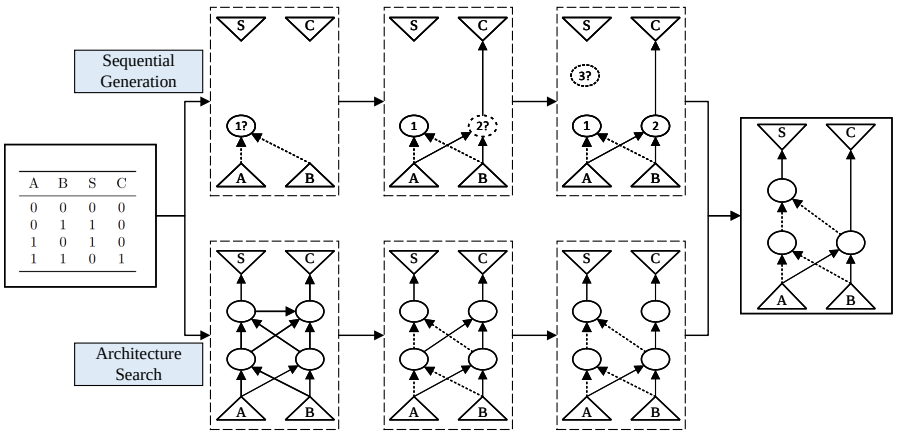

Fig. 3. Comparison between sequential generation and architecture search in logic learning.

Methodology. There are mainly two approaches to solve the problem, the details are in the following:

- **Differentiable Neural Architecture Search (DNAS)** transforms the traditionally discrete problem of neural network architecture search into a continuous, differentiable optimization problem, which enables the simultaneous optimization of both the network weights and the architectural parameters using gradient descent [93]. DNAS methods can be leveraged for generating circuit graphs by formulating the neural network as a circuit graph, where each neuron represents a logic gate and connections between neurons represent the wires connecting these gates. To facilitate differentiable training, [123] introduces continuous relaxation into discrete components in the neural network. First, the logical operations performed by logic gates are converted into their differentiable counterparts. For instance, $a\ AND\ b$ is relaxed to $a \cdot b$, and $NOT\ a$ is relaxed to $1 - a$ [123]. Second, the discrete network connections are parameterized, employing Gumbel-softmax [75] during forward propagation to continuously sample the connections between nodes. Then, the optimization process to explore a wide variety of connectivity patterns can be achieved by gradient descent.
  In the DNAS training process, the goal is to identify an optimal circuit structure, as shown in Fig. 3. The inputs of the network are the input vectors of AIG, and the outputs are the corresponding truth table vectors, while the labeled target is the ground-truth output truth table vectors. For example, when synthesizing an AIG, we denote the k-th neuron in the l-th layer by $o^{l,k}$. The neuron $o^{l,k}$, modeling an AND gate, has two inputs $c_0^{l,k}$ and $c_1^{l,k}$, representing either a normal or an inverted connection. Trainable parameters $W^{l,k} \in \mathbb{R}^{2\times\sum_{i=1}^{l-1} K_i}$ are associated with the wires to capture the connection choices for each neuron. Each element $w_{p,i,j}^{l,k}$ in the parameter represents the probability of connecting the j-th neuron in the i-th layer to the p-th input of the current neuron $o^{l,k}$. Another trainable parameter $I^{l,k} \in \mathbb{R}^2$, controls whether each connection is inverted. The computation of the forward propagation is as follows:

$$c_p^{l,k} = \sum_{i=1}^{l-1} \sum_{j=1}^{K_i} o^{i,j} \cdot \mathrm{softmax}(w_p^{l,k})_{i,j}, \quad p = 0, 1 \tag{1}$$

$$o^{l,k} = i_0^{l,k} c_0^{l,k} \cdot i_1^{l,k} c_1^{l,k} \tag{2}$$

$$\mathrm{Loss} = \sum_{j=1}^{K_L} l(o^{L,j}, TT[j]) \tag{3}$$

  where K_i denotes the number of neurons in i-th layer, TT is the truth table of dimension $K_L \times 2^n$, and the loss function l (e.g., MAE or MSE) is defined as the error between the network outputs and the truth table. Through backpropagation, the network gradually adjusts its parameters, converging toward a circuit structure that accurately reproduces the desired truth table outputs.
  The advantages of applying DNAS in LS are clear. By transforming the inherently discrete problem of circuit design into a continuous, differentiable one, DNAS enables the use of gradient-based optimization methods. This continuous formulation accelerates the search process, allowing for an efficient, gradient-based exploration of the high-dimensional solution space. In addition, DNAS offers a unified approach that provides an end-to-end framework, which eliminates the need for a separate design stage and simplifies the entire LS flow. However, this DNAS approach faces a critical scalability challenge regarding problem size. Because it relies on the full truth table for calculating the loss, its applicability is limited by the exponential $O(2^n)$ growth of the truth table with the number of primary inputs (n).

- **Sequential Generation.** Another popular ML roadmap models logic learning as the sequential generation process of the circuits, which construct node-by-node or edge-by-edge. By doing so, the generative methods are naturally applicable. One of the representative methods is the Autoregressive (AR) model [17], which leverages the chain rule of probability to factorize a joint distribution over previous variables. Specifically, the AR model decomposes the circuit generation process into a series of sequential decisions, where each decision determines the next element (node or edge) to add, conditioned on the current subgraph. The process is formally expressed as:

$$p(G^{\pi}) = \prod_{i=1}^{N} p(G_i^{\pi} | G_1^{\pi}, G_2^{\pi}, \cdots, G_{i-1}^{\pi}) \tag{4}$$

  where $p(G_i^{\pi} | G_1^{\pi}, G_2^{\pi}, \cdots, G_{i-1}^{\pi})$ represents the conditional probability of generating the i-th element given the previously generated elements, and π denotes a predetermined ordering for node generation in the graph.
  Despite their effectiveness in constructing circuits with correct structural connectivity, many of these methods tend to generate circuits with arbitrary or imprecise Boolean functions. Integrating truth-table information in the model training is crucial to address this shortcoming [194]. By incorporating truth tables as conditional inputs or node features, the action predictor can leverage latent representations drawn from a probability distribution to guide the generation process toward producing functionally accurate circuits. Moreover, truth tables can be directly used in the loss function to assess the correctness of the generated circuit truth table.
  However, the heuristic nature of ML generation methods inevitably leads to inaccurate graphs. As a result, additional fine-tuning is necessary, which is critical for achieving functional precision and structural correctness. Functional precision requires the synthesized circuit to strictly correspond to the truth table specifications, and structural correctness refers to adhering to the specific topological constraints of the target representation (e.g., an AND node must have exactly two inputs and one output), which a generic model might otherwise violate.

Representative Works. Based on the roadmap, recent ML works in Logic Learning can be categorized into the following two types.

- **DNAS.** The method in [13] first leverages DNAS for generating logic networks from simple truth tables. Their method utilizes universal logic units within the network architecture, connecting these units through residual combinations to facilitate cross-layer interactions. The architectural search focuses on selecting the appropriate unit for each layer, enabling efficient LS. The team from Google DeepMind in the IWLS 2023 contest employs DNAS for generating circuits from full truth tables [55] and achieved first place in the contest. Their approach demonstrates superior performance in terms of final circuit size. By using DNAS to generate novel circuit structures as a starting point, their method uncovered optimization paths that, after post-processing with traditional tools, led to results unattainable by traditional tools alone, highlighting the potential of DNAS in practical applications. The method in [162] proposes a regularized triangle-shaped circuit network with skip-connection in the DNAS model to address challenges such as excessive skip connections, inefficient searches, and imbalanced connections. The method in [165] combines the AR model with the DNAS model. The AR model is trained to generate the adjacency matrices representing circuit structures, which are then refined through the DNAS process to generate the circuit. The hybrid approach enhances the scalability and accuracy of logic circuit generation.

- **Sequential Generation.** ShortCircuit [155] integrates a Transformer-based network with an RL framework to learn generalizable patterns and efficiently explore the search space. The Transformer encodes the historical subgraphs and predicts the probability of node connections and the types of edges between nodes. The RL framework is employed to fine-tune the network on unseen designs, facilitating the discovery of near-optimal circuit architectures. Circuit Transformer [91, 92] proposes a sequential representation of circuits based on recursive replacement of wildcard nodes. It encodes the contextual information with a Transformer and employs Monte Carlo Tree Search (MCTS), which predicts the next symbol in the logic formula. This method transforms a non-optimal AIG into an improved, logically equivalent circuit. Boolformer [42] applies an embedding mechanism to map truth tables into a limited-dimensional space, feeding these embeddings into a Transformer to predict subsequent symbols in the circuit. SINE [161] brings symbolic regression into circuit generation by employing a modified MCTS to search for the next logic expression. The resulting expression is integrated with the current logic tree to construct the circuit. The method in [184] develops an adaptive decision tree model to predict the next logic gate in building the circuit. By learning from existing circuit structures, the decision tree can guide the generation process. LayerDAG [90] proposes a layer diffusion model to generate DAGs. This model systematically constructs the circuit layer by layer using an AR model, ensuring the acyclic property essential for logical circuit functionality. To further improve generation scalability, the method in [195] proposes a hierarchical conditional diffusion model for synthesis, which decomposes the generation process into hierarchical stages to effectively manage the structural complexity.

4 Boolean Circuit Analysis

The Boolean circuit analysis task primarily focuses on understanding the functional behavior of the Boolean circuits and extracting key features. This involves not only examining how the circuit processes inputs to produce outputs, but also delving into several critical areas, such as QoR prediction, SAT solving, DFT checking, and security. This section reviews the ML methods and applications in Boolean circuit analysis. Most of the ML methods apply GNNs in a supervised manner for circuit modeling, as the circuits are naturally represented as graphs. Additionally, traditional methods and other NNs are also applied in some tasks as the circuits can be represented in images or syntactic structures. Table 2 provides a detailed summary of the ML techniques and specific applications discussed in the following subsections.

4.1 Circuit Representation Learning

Circuit representation learning, which originates from graph representation learning, seeks to obtain the representation embeddings of the circuit components (gates and graphs). This emerging research field has significant potential for various applications in LS tasks.

The traditional representation approaches make use of the adjacency matrix or predefined features (e.g., number of gates, edges, and levels, etc.) to represent the circuits [70, 193]. Although such methods may lose some structural information of the circuits, they tend to be acceptable for simple tasks, such as serving as the state in an RL environment. However, for more complex tasks such as SAT solving and DFT analysis, these simple representations are less effective. In contrast, GNNs are then applied to extract features from the circuits to provide more informative representations for these tasks [28]. According to the tasks and purposes, the circuit representation learning methods are developed to perform various supervision tasks and downstream tasks.

Problem Statement. Let $G = (V, E, X)$ be the circuit graph, where V is the node set, E is the directed edge set, and X denotes the attributes of nodes and edges. The node embedding process

Table 2. Summary of ML Techniques and Applications in Boolean Circuit Analysis.

Domain	Task	Technique	Application	Type
Circuit Representation Learning (section 4.1) Node and Graph Classification	Logic Probability Prediction	MPNN [88] DAG-GNN [145]	SAT Sweeping SAT Solver	ML-Agent
		RL [141],DQN [141],ANN [74]	TPI	ML-Agent
		MPNN [8]	Reverse Engineering	ML-Agent
	Transition Probability Prediction	DAG-GNN [80, 81]	Power Estimation Testability Analysis	ML-Agent
	Structural Correlation Prediction	DAG-GNN,Transformer [145, 146, 192]	Circuit Classification	ML-Agent
	GraphSAGE [168]	Functional Reasoning	ML-Agent	
	Contrastive Pair Comparison	FGNN [158, 159]	Circuit Classification Subcircuit Classification	ML-Agent
	AIG Reconstruction	GNN [52, 144, 166], LLM [166]	QoR Prediction Logic Equivalence Identification	ML-Agent
SAT (section 4.2)	SAT or UNSAT Prediction	Ridge Regression [170] RF,DT,MLP,KNN,Bayes [40, 45] RNN [20, 138],GNN [11, 142],MPNN [22, 138] GAT [25, 26],Transformer [142]	SAT Solver	ML-Agent
	Unsatisfiable Core Prediction	MLP[137],Transformer[142] GNN[87, 142, 157, 178],RL[87]	CDCL Solver	ML-Assistance
	Solver Statistic Prediction	RNN [63],GNN [30, 188],RL [62]	Solver Enhancement	ML-Assistance
DFT (section 4.3)	Fault Diagnosing	Bayesian [72, 73],ANN [31]	Scan Chain Design	ML-Assistance
	Test Point Selection	MLP [105, 150],GCN [100, 163]	LBIST	ML-Assistance
	Output Prediction	ANN [131–133],PCA [131],GNN [190]	ATPG	ML-Assistance
Security (section 4.4)	Structural Change Prediction	Random Forest [24],GNN [140]	Obfuscation Attack	ML-Agent
	Decryption Key Finding	Genetic Algorithm [29],GNN [6–8, 69, 101]	Logic Locking Attack	ML-Agent
	Leakage Detection	GNN [41, 175]	Defending	ML-Agent

learns a mapping function $f_v : (v_i, G) \rightarrow \mathcal{R}^d$ that encodes node v_i into a d-dimensional vector. This mapping is designed so that the similarity between nodes in the graph is preserved in the embedding space. Similarly, graph embedding learns a mapping function $f_G : G \rightarrow \mathcal{R}^d$ that encodes the entire graph G into a vector, ensuring that similarities between different graphs are maintained within the embedding space. In the ML-Agent method, these circuit embedding vectors are then processed by a decoder to accomplish the metrics prediction or other supervision and downstream tasks. The flow of the circuits representation learning is shown in Fig. 4.

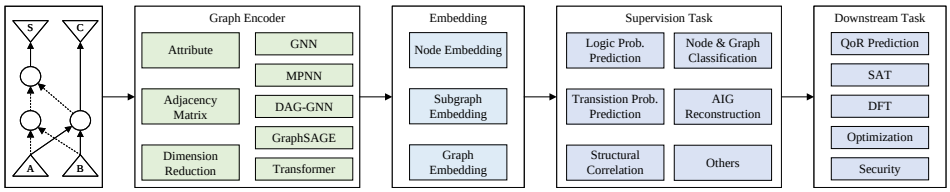

Fig. 4. A general flow of circuit representation learning.

Circuit Embedding. GNNs are the most common architecture for this task. A GNN learns a node embedding h_i by iteratively aggregating information from its neighbors $\mathcal{N}(i)$ and combining it with the node's previous state. This general message-passing process is often formulated as:

$$h_i^{t+1} = f(h_i^t, \{h_j^t, \forall j \in \mathcal{N}(i)\}) \tag{5}$$

In matrix form, this update can be written as:

$$H^{t+1} = F(H^t, A, X) \tag{6}$$

where H^t is the node embedding matrix at iteration t, X is the initial node feature matrix (derived from G's attributes), and A is the adjacency matrix. After T iterations, the final embeddings are used for further exploration.

Several popular GNN-based circuit embedding methods are presented below:

- **GraphSAGE** employs a sampling and aggregation strategy for inductive node embedding [61]. It leverages node features, such as gate types and inverter connections [168], to capture richer structural and functional information. In this approach, GraphSAGE learns a set of aggregation functions that combine features from the local neighborhood of each node and then propagates the aggregated information to the target node v_i. Then the hidden state of node v_i at layer l is updated as follows :

$$h^l_{\mathcal{N}(i)} = \text{AGGREGATE}_l(\{h^{l-1}_j | \forall j \in \mathcal{N}(i)\}) \tag{7}$$

$$h^l_i = \sigma(\mathbf{W}^l \cdot \text{CONCAT}(h^{l-1}_i, h^l_{\mathcal{N}(i)})) \tag{8}$$

  where h^0_i denotes the initial node attributes and AGGREGATE represents the differentiable aggregator function. Common functions include mean aggregator, pooling aggregator, or Multilayer Perceptrons (MLP)-based aggregator [61]. The subscript l in AGGREGATE_l signifies that the differentiable aggregation function is specific to layer l, as the model can learn a distinct aggregator for each layer. $\mathcal{N}$ is the neighborhood function (typically the set of 1-hop topological neighbors [168]), $\mathbf{W}^l$ denotes the weight matrices, and σ is a non-linear activation function. After L layers of propagation, the embedding of each node encodes both structural and functional information from its neighborhood.
- **DeepGate** employs the Message Passing Neural Network (MPNN) architecture [53], integrated with DAG-GNN [153] to capture both functional and structural information of the circuits, which performs the message propagation in a topological order between nodes and only considers the predecessors in the AGGREGATE operation. Furthermore, Deepgate applies the same model recurrently for T times to generate the final embedding, as described by

$$h^l_i = \text{COMBINE}_l(h^{l-1}_i, \text{AGGREGATE}_l(\{h^l_j | \forall j \in \mathcal{P}(i)\})) \tag{9}$$

$$h^t_i = \text{COMBINE}(h^{t-1}_i, \text{AGGREGATE}(\{h^t_j | \forall j \in \mathcal{P}(i)\})) \tag{10}$$

  where $\mathcal{P}(i)$ denotes the set of predecessor nodes (direct fan-ins) of node i. The COMBINE (or layer-specific COMBINE_l) function is a differentiable update function that merges the node's previous state with the newly aggregated message from its predecessors.The major difference between Eq. 7 and Eq. 9 is that in DAG-GNN, the aggregation for node v_i will be only executed after all of its predecessor hidden states have already been computed. Moreover, DeepGate incorporates an attention-based aggregation mechanism [156] and employs a Gated Recurrent Unit (GRU) [185] as the update function, which are formulated by

$$m^t_i = \sum_{j \in \mathcal{P}(i)} \alpha^t_{ij} h^t_j \quad \text{where} \quad \alpha^t_{ij} = \underset{j \in \mathcal{P}(i)}{softmax}(w_1^\top h^{t-1}_i + w_2^\top h^t_j) \tag{11}$$

$$h^t_i = GRU([m^t_i, x_i], h^{t-1}_i) \tag{12}$$

  where w_1 and w_2 are learnable weight vectors of the attention mechanism, t represents the current recurrent iteration step, α^t_{ij} denotes the attention coefficient, m^t_i is the aggregated message and x_i is the initial feature. By introducing attention and the GRU mechanism, Deepgate effectively captures context-aware features and accelerates the convergence of the training process.

- **FGNN** aims to distinguish functionally equivalent and inequivalent netlists via CL [159]. Similar to general circuit representation learning, netlist representation is also built upon the DAG-GNN architecture to generate embeddings. The key innovation of FGNN is the design of its CL loss function. Positive samples are generated by applying the Boolean equivalent but topologically distinct transformations to subnetlists. This procedure ensures the functional behavior remains consistent while the structural form changes, making the contrastive objective clear and explicit. Negative samples are the netlists with different functionality and are much more than the positive samples. To encourage the model to learn functionally discriminative features, a contrastive loss is used to minimize the feature distance between positive pairs while maximizing the distance between negative examples. The loss is defined as:

$$L = \frac{1}{N}\sum_{n=1}^{N} l_{1,2}(n) + l_{2,1}(n), \quad \text{where} \tag{13}$$

$$l_{i,j}(n) = -\log \frac{\exp(\mathrm{CST}(h_{n,i}, h_{n,j}))}{\sum\limits_{u=1,u\neq n}^{N} \mathrm{CST}(h_{n,i}, h_{u,i}) + \sum\limits_{u=1}^{N} \mathrm{CST}(h_{n,i}, h_{u,j})} \tag{14}$$

$$\mathrm{CST}(h_{n,i}, h_{n,j}) = \frac{h_{n,i}^{\top} h_{n,j}}{||h_{n,i}||\,||h_{n,j}|| \cdot \tau} \tag{15}$$

  where $h_{n,i}$ denotes the embedding of the i-th augmentation of n-th netlist, and CST represents the cosine similarity between two embeddings scaled by a temperature parameter τ. The CL framework [159] generates one positive pair of augmented netlists $(h_{n,1}, h_{n,2})$ per sample n. Eq. 13 thus calculates the average symmetric loss $(l_{1,2} + l_{2,1})$ for all N pairs in the mini-batch.

Supervision Tasks and Downstream Tasks. Although GNN can generate feature embeddings for circuits, these embeddings differ significantly according to different tasks. The supervision tasks are specifically designed to help GNNs develop a deeper understanding of the circuit functionality and structure by deciding their convergence directions and the generalization ability according to the specific task. In contrast, downstream tasks, while ideally correlated with these supervision tasks, are challenging to integrate directly into the model training process. Furthermore, downstream tasks are more closely aligned with the LS flow, including applications such as SAT solving, DFT analysis, and other tasks.

The supervision tasks involve predicting circuit behavior, assessing performance metrics, or comparing circuits against negative samples. Here are some examples:

- **Truth Table-based Functionality Prediction.** DeepGate [88] first proposes the logic probability prediction task, which estimates the probability of being logic '1' for every node. Based on this, DeepGate2 [145] evaluates the logic discrepancies of node pairs by comparing the predicted truth tables.
- **Transition Probability Prediction.** DeepSeq [80] introduces the prediction of transition probability, which is the frequency of the node signal $0 \rightarrow 1$ and $1 \rightarrow 0$ transition. DeepSeq2 [81] employs the pairwise Flip-Flop (FF) similarity, which assesses the occurrence of identical state transitions for FF pairs under the same input conditions.
- **Structural Correlation.** DeepGate2 [145] also proposes a loss function to indicate whether a node pair has a common predecessor to embrace structural information. In DeepGate3 [146], metrics such as graph edit distance, logic levels, pair-wise connections, and more are computed for loss convergence.
- **Node and Graph Classification.** Gamora [168] classifies the nodes by their functionality, such as XOR, MAJ, etc. ConVERTS [32] and the method in [50] use the functional similarities

of circuits as supervision. Then ConVERTS [32] utilizes the clustering method to classify the representative embeddings into different classes.

- **AIG Reconstruction.** MGVGA [166] applies a masked autoencoder and selectively obscures portions of the graph, such as node features or edges, through a masking operation. The AIG reconstruction is achieved by the prediction of gate types and gate-level degrees. In addition, GNP [52] predicts the reconstructed adjusted adjacency matrix, and DeepCell [144] reconstructs the masked netlist by predicting the logic-1 probability and functional similarities.
- **Contrastive Pairs Comparison.** FGNN [159] designs some Boolean equivalent pairs manually in the dataset and compares the similarities of these pairs. FGNN2 [158] also compares pairs, but specifically compares the similarity of their corresponding truth table vectors. Both tasks are unsupervised without labeled data.

Downstream tasks aim to utilize the representation models to enhance performance in specific applications, including:

- **SAT Sweeping.** DeepGate2 [145] integrates the model into the SAT sweeper to guide the equivalence class selection task, and a SAT solver to solve the instances from the logic equivalence checking task.
- **SAT Solving.** DeepSAT [89] formulates SAT solving as a conditional generative procedure constructs more informative supervisions with simulated logic probabilities. The logic probability of PO is approximated to solve the SAT problem.
- **QoR Prediction.** The DeepSeq family [80, 81] employs the dynamic power estimation for sequential circuits as the prediction problem. Circuit-GNN [173] employs routing congestion prediction and net wirelength prediction in placement as downstream tasks. GNP [52] predicts the QoR as the downstream task.
- **DFT.** The method in [74] presents an ANN to predict the logic probability and applies the ANN to test point insertion (TPI) to improve fault coverage. DeepTPI [141] formulates the TPI as an RL problem and finds the optimal test point via the combination of a trained deep Q-learning network and the DeepGate model.
- **Reverse Engineering.** The method in [66] learns the representation of circuits to predict the boundary of arithmetic blocks in the circuit. The method in [189] detects functional operators and applies reverse engineering on the netlist to generate the original RTL. Gamora [168] takes the node classification as supervision and couples the classification result into the symbolic reasoning task.
- **Technology Mapping.** DeepCell[144] integrates multiview information from both AIGs and Post-Mapping netlists. It is primarily used to enhance the efficiency and performance of two critical EDA tasks: functional Engineering Change Orders (ECOs) and TM.

However, in addition to utilizing circuit representation learning methods to accomplish these downstream tasks, researchers have developed some unique ML-based methods for each task. Some are ML-Agent methods that directly supervise the required metrics, and others are ML-Assistance methods that use statistics produced by traditional algorithms as supervision. In the following sections, we will briefly review the ML methods and applications in different tasks.

4.2 Boolean Satisfiability

The Boolean SAT problem is reviewed here because its solvers serve as a core computational engine for many LS tasks, including equivalence checking, SAT sweeping, and SAT-based exact synthesis. The Boolean SAT problem asks: Given a Boolean formula, does there exist an assignment to each variable such that the formula is TRUE? The LS technology can be applied to the SAT solver

by reconstructing the instances to improve the internal structure and logical representation of the circuits, transforming the original SAT into a more easily solvable version. Additionally, ML methods provide attractive solutions to SAT problems, as surveyed in [57, 68].

The standalone ML-based SAT solvers can predict the SAT and decode assignments with end-to-end deep learning models solely. NeuroSAT[138] is a pioneering model that first introduces learning an SAT solver from single-bit supervision and utilizing both RNN and GNN to process the CNF formula for SAT classification, but it leads to failure because of the dataset imbalance [30]. By modeling SAT formulas as literal-clause graphs and employing MPNNs, NeuroSAT effectively captures structural relationships within SAT formulas. However, its scalability diminishes notably when applied to larger and more complex problem instances. Similar works that give the direct prediction or the indirect probability for SAT problem include [11, 22, 25, 26, 142]. Furthermore, Query-SAT [118] introduced a query mechanism and unsupervised loss functions, enabling iterative refinement of SAT-solving strategies by reducing the search space and enhancing reasoning efficiency. These studies provide insights into the strengths and limitations of GNNs in SAT-solving, paving the way for future advancements in the field.

On the other hand, ML-Assistance methods that combine ML models with modern SAT solvers have also emerged as a promising research direction. These methods improve traditional SAT solvers by leveraging NN predictions to develop heuristics, significantly enhancing the performance of Conflict-Driven Clause Learning (CDCL) solvers and local search algorithms [103]. For instance, NeuroCore [137] predicts unsatisfiable cores, enabling periodic adjustments to variable activity scores in CDCL solvers to prioritize challenging instances. Graph-Q-SAT [87] combines RL with GNNs to optimize branching decisions in SAT solvers. By learning effective branching heuristics, Graph-Q-SAT reduces the number of iterations required to solve SAT instances, thereby improving solver efficiency. Similarly, the method in [178] trains a GNN with RL to learn a local-search heuristic that finds a solution by flipping a single variable in each step. The GVE algorithm integrates glue variable elimination with graph learning to simplify formulas and enhance efficiency [157]. Learning cubing heuristics from DRAT proofs improves variable selection in the cube-and-conquer framework [63]. It is also theoretically shown that GNN can learn to mimic the WalkSAT algorithm [30]. Additionally, learned heuristics improve local search methods by refining diversification and intensification strategies, enabling solvers to tackle more complex SAT instances [178, 186]. These approaches broaden the adaptability of traditional solvers, enabling them to handle a wider variety of SAT challenges.

It is worth noting that although ML-Agent and ML-Assistance methods show improvements in search efficiency and effectiveness, these methods fall short in applicability to industrial production of large instances and suffer from high computational cost. Moreover, the intrinsic explainability problem of NNs challenges the trust and safety, which requires additional proof and verification.

4.3 Design For Testability

The DFT technique refers to the set of design techniques and methodologies integrated into the LS process so that the final design is more amenable to testing. The techniques during DFT in LS include: scan chain design, Logic Built-In Self-Test (LBIST) [47], Automatic Test Pattern Generation (ATPG), etc. A detailed survey of ML in DFT can be found in [134]. Here, we list some ML-based works in DFT of the circuits.

Scan Chain Design. A scan chain is a "shiftable" chain formed by concatenating registers of signal "$0, 1, X$" in a specific order, used to inject test data into circuits and read responses to test whether the internal logic is normal. The methods in [72, 73] introduce Bayesian learning in an unsupervised manner for identifying faulty scan cells, which shows positive results in diagnosing designs containing intermittent faults. Another work proposes an Artificial Neural Network (ANN)

in a supervised manner to diagnose faults in the scan chain by inputting the fault features and directly outputting the scan cells [31].

LBIST and TPI. To improve the fault coverage of LBIST, TPI techniques find high-quality TPs to improve fault coverage or reduce test vector count. The method in [105, 150] uses a fully-connected NN to evaluate the impact of control-0 and control-1, and predicts the impact of TPs on the stuck-at-fault coverage for pseudo-random stimulus, which improves the fault coverage and significantly reduces TPI time. The method in [100] uses a Graph Convolutional Network (GCN) to classify signal nodes as either easy-to-observe or difficult-to-observe. The method predicts the observation of TPI for each node and its neighbors, improving the TPI efficiency. The method in [151] proves that increasing the NN complexity can improve the LBIST performance with higher fault coverage and fewer test points while reducing the computation time. The method in [163] uses GCN trained with RL to identify the optimal combination of test points that maximizes the random testability of large-scale logic circuits.

ATPG. The application of ML in ATPG is related to the heuristic part of the ATPG algorithm [134]. All programmed algorithms have used heuristics to speed up the search, and ML techniques can accelerate the generation process. The method in [131] uses an ANN and PCA to distill circuit topological information and testability to guide the heuristic search. Methods in [132, 133] supervise circuit data to enhance ATPG and reduce backtracks by replacing conventional heuristics to decide backtracing direction using an NN trained with data on hard-to-detect faults. The method in [190] introduces a GNN-based autoencoder to predict hard-to-detect faults in ATPG preprocessing.

4.4 Security

Circuit security mainly refers to protecting circuit designs and hardware Intellectual Property (IP) from the risk of unauthorized copying, theft, or overproduction. Common protection measures include logic locking and obfuscation. Most existing ML-based works are more interested in attacking the circuits, and only a few works focus on providing a defense to secure their ICs.

For **obfuscation attacks**, SAIL [24] utilizes a random forest ML model to retrieve the design of obfuscated circuits and predict the transformations in circuit topology made by obfuscation methods. CUTSAIL [140] applies the GNN on the adjacency matrices of circuits to predict their functionality and the missing k-cut parts of the obfuscated circuits. UNTANGLE [5] formulates the key-extraction task as a link prediction problem and trains a GNN to predict the links that can be used for attacking the obfuscating blocks.

For **logic locking attacks**, GALU [29] first introduces the Genetic Algorithm (GA)-based logic unlocking attack by gradually searching the decryption key in the key space. OMLA [7] employs the GNN-based subgraph classification to find the key-bit values of the key gates to attack logic locking. MuxLink [6] extracts structural and functional hints from the subgraph and uses GNN to perform binary classification on the extracted subgraph to predict the correct links. The method in INSIGHT [101] introduces an explainability-guided GNN attacker that recovers secret keys from existing locking techniques. The method in [69] formulates the logic de-camouflaging problem as the node classification problem and proposes a GNN that aggregates features from fan-in and fan-out nodes to classify the camouflaged nodes. The method in [8] focuses on reverse engineering, which refers to identifying different parts of circuits with the intent of duplicating them. By leveraging the GNN, the method can convert circuits back to different modules, such as adders, multipliers, control logic, etc.

For **defending** these attacks, Flip-Lock [41] first develops its own ML-based analysis tool to identify potential structural leakages near FFs. In addition, it uses this trained ML model within its design flow to evaluate and strengthen the resilience of the final Flip-lock design, ensuring the attacker's accuracy is reduced to near random guessing. GNN4IP [175] proposes a GNN-based IP

piracy detection technique that evaluates the similarity between two circuits to identify the pirated circuit.

5 Logic Optimization

The AIG has become a cornerstone in LO, largely due to its common implementation using structural hashing [108]. This technique ensures that identical logic subgraphs are uniquely represented and reused automatically. This automation simplifies algorithm development by minimizing the need for manual tuning or trial-and-error within algorithms to find and merge redundant logic. Additionally, reductions in AIG node count and logic levels provide a clear, quantitative measurement of improvement. This metric aligns well with both standard-cell and Field-Programmable Gate Array (FPGA) mappers, which typically use AIGs or similar graph structures [107]. AIG also supports a variety of structural optimizations, such as rewriting, balancing, and refactoring.

Given a Boolean network, designers must arrange these subgraph heuristics in a carefully chosen sequence to achieve the best overall results. The process is a complex sequential decision-making task in which these optimization algorithms interact with each other to produce an optimized gate-level implementation.

Researchers traditionally rely on years of synthesis experience to craft effective transformation sequences for new IPs. However, as designs grow in complexity, manual adjustments are increasingly too slow and inefficient to generate optimal netlists. ML-based approaches are emerging to automate and accelerate this process. Key ML tasks in LO include:

- **QoR Prediction.** The foundational ML task is to estimate the QoR for a given design and recipe. This capability not only speeds up evaluation but also supports other ML research efforts. Variants include predicting QoR for: (a) Seen IPs with unseen recipes, (b) unseen IPs with seen recipes, and (c) unseen IPs with unseen recipes [34].
- **Boolean Representation Classification.** Different optimizations are most effective on specific graph representations (e.g., AIG, MIG, XMG). Classifying which representation best suits a subcircuit enables the selection of the most appropriate optimization algorithms, improving the final QoR [110].
- **Recipe Exploration.** One roadmap formulates recipe search as an MDP, where the future AIG depends on the current AIG (state) and synthesis transformation (action). RL agents sequentially generate recipes to maximize QoR. An alternative roadmap is the Design Space Exploration (DSE) approach, which evaluates the recipes holistically, often via heuristic sampling or evolutionary strategies [56, 59].
- **Algorithm Improvement.** Since traditional heuristics cannot guarantee optimality, ML models are integrated directly into synthesis algorithms, either to guide heuristic choices or to propose entirely new transformations [121].

In the following sections, we will introduce ML methods and applications for each task.

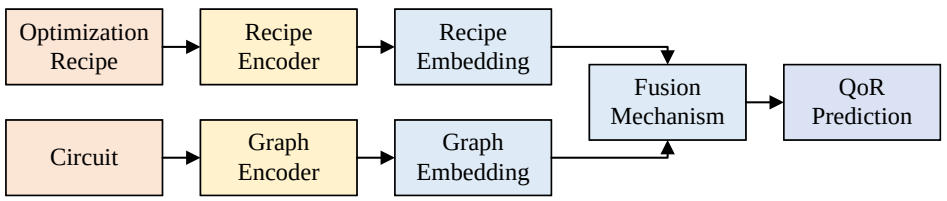

Fig. 5. Flow of QoR prediction process.

5.1 QoR Prediction

Determining the optimal synthesis recipes in LO is a complex challenge. Empirical observations indicate that no single synthesis strategy performs optimally across all IPs. Because each synthesis run can be computationally expensive, accurately predicting the QoR for a given IP-recipe pair helps designers quickly identify the most promising strategies. For example, using the normalized number of AIG nodes remaining after synthesis as the QoR metric provides a clear, quantitative basis for comparing different recipes.

Beyond its direct value, QoR prediction can support many other tasks in LS. In recipe exploration, predicted QoR scores can guide the search process, either by getting quicker rewards back for the agents or by ranking candidate recipes in advance in DSE. While traditional synthesis tools can estimate QoR, ML models offer far greater efficiency.

Most state-of-the-art works encode circuit structure and recipe sequences separately, usually use graph neural networks for the former and sequence models for the latter, and then jointly predict the expected QoR for any unseen recipe–IP combination. A general process for QoR prediction is illustrated in Fig. 5.

Problem Statement. Let $G = (V, E, X)$ denote the circuit graph and $O = \{o_1, o_2, \cdots, o_n\}$ denote the set of available subgraph transformations.A recipe r is an ordered sequence of operations drawn from O. Given a circuit G and a recipe r, the TM stage produces a mapped netlist on which the QoR can be measured by standard EDA tools. We denote the measured QoR by $y = QoR(G, r)$, which serves as the ground truth label during model training. In the inference phase, the goal is to predict y from the pair (G, r), enabling rapid estimation of QoR without performing full synthesis.

Formally, the QoR prediction task is to learn a mapping function f that takes a circuit-recipe pair (G_i, r_j) as input and outputs a numerical value $\hat{y}_{ij}$, which represents the predicted QoR. We train f by minimizing a loss function, such as the common Mean Squared Error (MSE), between predictions and ground truth:

$$\hat{y}_{ij} = f(G_i, r_j) \tag{16}$$

$$loss = \sum_i \sum_j (y_{ij} - \hat{y}_{ij})^2 \tag{17}$$

In this way, f learns to approximate the expensive synthesis process, providing fast and accurate QoR estimates for unseen circuits and recipes.

Methodology. Most QoR prediction methods adopt an encoder–predictor pipeline comprising four key components: a graph encoder, a recipe encoder, a fusion mechanism, and a predictor.

- **Graph Encoder.** For the AIG encoder, GNNs (as generally defined in Section 4.1, Eqs. 5 and 6) are a widely used architecture. The GNN variants frequently employed in QoR prediction include the GCN [82] and the Graph Isomorphism Network (GIN) [169]. These models represent different design trade-offs: GCN is often valued as an efficient and simple baseline, while GIN is chosen for its maximum theoretical expressive power. The GCN [82] architecture provides a localized first-order approximation of spectral graph convolutions with a layer-specific trainable weight matrix that performs the feature transformation. Conversely, GIN [169] is designed to be as powerful as the Weisfeiler-Lehman graph isomorphism test.
- **Recipe Encoder.** Recipes are the ordered sequences of subgraph transformations. These transformations can be naturally represented as one-hot vectors since they are discrete and uncorrelated. Therefore, the recipes can be encoded numerically by one-hot matrices. These matrices are then projected into a dense embedding space via a learnable weight matrix. To capture sequential patterns, the embedded recipe can be processed by: (1) 1D convolution layers [34], (2) Recurrent Networks such as Long Short-Term Memory (LSTM) [167], or (3)

Transformer encoders [171]. Furthermore, alternative encodings, such as randomly initialized embeddings or heuristic-based vectors, have also been explored.

- **Fusion Mechanism.** A straightforward fusion is to concatenate the graph and recipe embeddings. However, simple concatenation may underutilize cross-modal interactions, especially when embedding dimensions differ [149]. To better capture complex, non-linear relationships between graph embeddings and recipe embeddings, more advanced fusion strategies are used: (1) graph pooling [46] can harmonize embedding sizes before fusion, (2) attention mechanism [77, 171] can learn to weight graph versus recipe features dynamically, or (3) hybrid model interaction [167], to better comprehend the complementary information between circuits and recipes.
- **Predictor.** Finally, the fused embedding is decoded to a QoR estimate. Common predictors include: (1) MLPs are the simple and efficient decoders for regression, (2) the Transformer decoder is used when leveraging cross-attention between graph and recipe embeddings, and (3) Binary Classifiers [180] predict whether the recipe is good or bad for a given circuit rather than producing a numerical score.

Table 3. Summary of ML Methods in QoR Prediction.

Method	Input	Technique	Predicted QoR	Highlight
OpenABC-D[34]	AIG + Recipe	GCN	Area & Delay	Public dataset
[181]	Recipe	LSTM	Area & Delay	Transfer learning on physical design
GRANNITE[187]	Netlist + Feature data	MP-GNN	Power	GPU acceleration
LOSTIN[167]	Circuit + Recipe	GNN + LSTM	Area & Delay	Exploiting spatio-temporal info GNN with supernode
[171]	Circuit + Recipe	GNN + Transformer	Area & Delay	Applying Transformer to extract feature from recipes Experiments on various methods
Bulls-Eye[12]	AIG + Recipe	GCN + 1D-CNN	#Nodes & Area	Few-shot learning
LSOformer[77]	AIG + Recipe set	GCN + Transformer	Trajectory of Area & Delay	Cross-attention fusion of graph and recipe embeddings
LSTP[191]	AIG + Recipe	GNN + Transformer	Delay	RTL-analyzer
AiLO[21]	AIG + Recipe	GNN + Transformer	Area & Delay	Multi-scale cross-attention fusion; Ranking preservation
CongestionNet[83], CeConP[15]	Netlist Node centrality, Cell feature[15]	GAT	Congestion	Prediction of physical metrics
[71]	Netlist	Decision Tree	Delay	Guiding optimizations in earlier logic synthesis

Representative Works. Recently, there has been a considerable amount of work on the QoR prediction tasks. Here we survey some of the representative works, as listed in Table 3. OpenABC-D [34] employs the GCN layer on each node's embeddings and aggregates them by both max and mean readout functions to get the graph embeddings. It also experiments on the three variants of the tasks and provides a baseline for research. The method in [181] proposes an LSTM regression architecture to encode the recipe as a time-ordered sequence of optimizations, regressing directly to the QoR. Moreover, it shows the cross-design transfer learning by pretraining on large data and fine-tuning on small data. GRANNITE [187] introduces a GNN-based power estimation model whose key feature is transferability. After training on one set of designs, it can directly infer the average power of new, unseen netlists without retraining, thereby accelerating power analysis flows that traditionally require days of simulation. LOSTIN [167] jointly encodes the netlist with a GNN (spatial) and the recipe with an LSTM (temporal), then concatenates the spatial and temporal embeddings for QoR prediction. In order to learn the "spatial information" of AIG graphs, this article explicitly explores GCN and GIN as architectural options for its GNN components. In addition, the recipes are embedded into a supernode and added to the circuit for better fusion. The method in [171] combines a GNN with a Transformer under a joint learning policy to establish the correlation between the circuit structure and the recipe language feature. Similarly, AiLO[21] introduces a

Table 4. Comparison of ML Methods in QoR Prediction and Their Practical Utility Analysis

Core ML Method	Reported Metrics	Practical Utility Analysis
Concatenating AIG (GCN) + recipe (1D-CNN) encodings for prediction [34].	MSE = 0.536 ±0.03 on unseen IP-recipe.	Providing a baseline model allowing for basic screening.
Model fine-tuning using transfer learning (14 nm → 7 nm) [181].	MAPE ≤ 3.7% on unseen recipe and library.	Highly practical. Designers can run a few recipes to fine-tune an accurate screener.
Concatenating spatial and temporal encoders for prediction [167].	MAPE ≤ 3.1% on unseen designs.	Providing accurate, "off-the-shelf" recipe screener without re-training.
Concatenating AIG (GCN) + recipe (Transformer) encodings [171]. [77] applies cross-attention for fusion.	MAE = 0.412 on unseen circuit-recipe [171]. MAPE: 4.35%-17.06% improvement on unseen circuit [77].	The Transformer can use long-range dependencies to find complex, non-linear "good" recipes.
Few-Shot Active Learning for model fine-tuning [12].	Generating recipes of 95% quality on unseen designs with a 2x-30x speedup.	Operating within a limited budget by running "most informative" recipes to rapidly find solutions.
Multi-scale cross-attention mechanism to fuse graph and recipe embeddings [21].	Top-10% ranking accuracy: 40.46%; Avg. improvement 42.5%	High ranking fidelity. The model prioritizes maintaining the relative order of recipes.

predictive framework utilizing a multi-scale cross-attention Transformer to effectively capture long-range dependencies and structural features for accurate QoR prediction. Bulls-Eye [12] employs few-shot transfer learning and active sampling to adapt QoR prediction models to unseen IPs with minimal additional data. The method also makes experiments on different combinations of circuit encoders and recipe encoders, showing that the combination of GraphSAGE and Transformer gives the best results. LSOformer [77] applies the level-wise pooling to convert the DAG to the circuit embeddings and then applies the cross-attention fusion for circuit and recipe embeddings, followed by a Transformer decoder to predict the trajectory of QoRs. LSTP [191] introduces a hierarchical cone sampling strategy and a specialized GNN to learn the intrinsic features of AIG. Methods in [15, 83] apply GAT on gate-level netlists to predict the routing congestion in physical design before placement. Similarly, the method in [71] utilizes a gradient boosting decision tree [79] to predict the post-technology-mapping delay based on pre-mapped gate-level netlists, allowing it to guide optimizations in earlier synthesis.

To provide a clearer comparative analysis for the specific task of recipe screening, Table 4 provides the practical utility of each architectural approach, which highlights the characteristics that a designer must consider when selecting a model to efficiently find high-quality synthesis recipes.

5.2 Boolean Representation Classification

The motivation for Boolean representation classification arises from both the divide-and-conquer strategy and the TM stage, in which each k-bounded subgraph of the circuit is sequentially matched to a physical cell. Because different optimizations perform best on different graph representations, it becomes advantageous to partition the circuit into subgraphs and select the most suitable Boolean representation for each one. Concretely, this task seeks to learn a classifier that, given a subgraph extracted from the circuit, predicts which representation, such as AIG, MIG, or XMG, will produce the highest QoR under each synthesis recipe.

Problem Statement. Given a Boolean network represented as a DAG, the first step is to decompose the DAG into smaller subnetworks (partitions). Each partition is handled as a separate module and should determine the most appropriate Boolean representation. Then, the best synthesis recipes suited to the predicted representation are applied to each partition to perform local optimizations independently. After that, partitions are merged back into the original DAG structure and ensure the functional correctness of the recomposed circuit. This process flow is illustrated in Fig. 6.

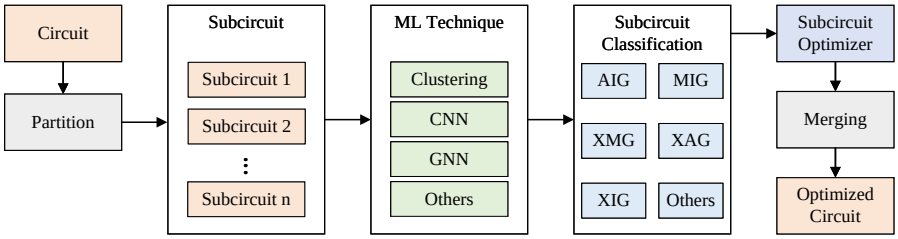

Fig. 6. Flow of Boolean classification process.

Methodology. Similar to the QoR prediction problem, Boolean representation classification also relies on informative graph embeddings. However, models trained on full circuits cannot be directly applied to subgraphs due to structural and functional domain gaps. Some recent methods address this challenge.

In LSOracle [110], each partitioned subcircuit is small enough to be represented as a moderate Karnaugh map [78], where the input combinations that evaluate the function to logic true are grey pixels, while input combinations that evaluate the function to logic false are denoted by black pixels. Then this image is fed into an MLP trained to predict the optimal Boolean representation (e.g., AIG and MIG). By converting Boolean functions into a visual format, LSOracle bridges the gap between circuit structure and learnable features. In HeLO [124], the partitioned circuits are encoded via a GNN to generate embedding vectors that capture local topology. Then these embeddings are agglomeratively clustered in the feature space. Each cluster corresponds to a recommended Boolean representation, as determined by cluster centroids and GNN predictions. For very small partitions, the clustering process excludes them and instead receives a default or independently learned optimization flow.

5.3 Recipe Exploration

The exploration of synthesis recipes can be broadly categorized into two paradigms based on the underlying generation patterns: (1) Sequential Generation via RL. This approach models the recipe exploration as an MDP, where an RL agent builds a recipe step-by-step. At each step, the agent selects the next transformation based on a reward signal reflecting the current QoR. This MDP allows dynamic adjustments, enabling adaptive refinement of the synthesis strategy. (2) Holistic Exploration via DSE. In contrast, DSE methods take a comprehensive view of the entire synthesis recipe space. Instead of sequential construction, DSE systematically evaluates diverse transformation combinations, often using heuristics or sampling. This method aims to identify promising recipes across possibilities, potentially uncovering high-performing solutions missed by sequential approaches. Both paradigms offer distinct advantages. RL-based sequential generation excels at adaptive fine-tuning using immediate performance feedback, while DSE provides a broader search that can reveal novel and effective strategies across the solution space.

Problem Statement. Given an AIG representing Boolean functionalities, the problem is to determine the sequence of optimization steps that yields a functionally equivalent AIG with optimal characteristics (e.g., minimal size or depth). The timing complexity of the problem is $\sum_p^2$-Hard [34].

RL Methods. To apply the RL framework for recipe exploration in LO, the task is formulated as an MDP problem. The core components of this MDP are:

- State: A state s consists of the statistical information from the current Boolean circuit and other useful information from the history experience, to represent the feature of the environment at a specific timestep

- Action: An action a performs a synthesis optimization provided in the synthesis tool and changes the status of the DAG. The action space is a set of valid actions that can be taken for the Boolean circuit.
- Reward: After finishing the action, the environment will reach the new state, and give the reward to the agent as the feedback. The reward r is defined to indicate the improvement on the final objective.
- State transition: The process of moving from one state to another is called transition. The transition probability describes the probability distribution over the next states.

The RL agent sequentially executes an action at a time. This process is shown as:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots s_{n-1} \xrightarrow{a_{n-1}} s_n \tag{18}$$

where s_0 refers to the state of original Boolean network, s_i refers to the $i-$th step of the circuit state. Given an action a_i, the agent will interact with the LS environment and observe the reward.

During the process of exploring the environment, the agent interacts with the environment in a continuous way until reaching the terminal state. The interaction experience from the initial state to the terminal state constitutes the trajectory in an episode. For a single episode, the series of chosen actions performed by the agent constitutes a synthesis script.

Based on the modeling of the synthesis tasks, the learning framework for exploring the design space of LS is presented in Fig. 7. The framework combines two major parts: the RL agent and the LO environment. For the LO environment, the Boolean network, technology library, and design constraint are used as inputs and sent into the LS tool, such as ABC [19]. The next state and reward are then sent back to the RL agent. For the RL agent, the state and reward are used to guide the training process of the neural network, which outputs the next action based on a value-based or policy-based strategy.

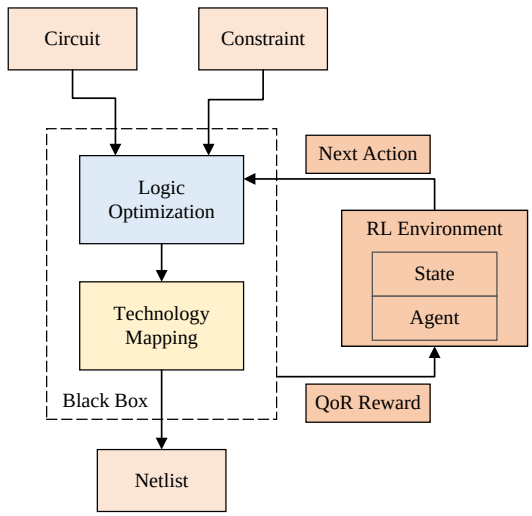

Fig. 7. The RL framework of the recipe exploration task.

The basic principle of the agent is to learn a policy that maximizes the expected long-term reward. This long-term reward, formally known as the return R_t at a given time step t, is defined as the total discounted accumulated reward from that step until the end of the episode at step n. It

Table 5. RL vs. DSE Comparison

Evaluation Metric	RL	DSE
Core Paradigm	Sequential decision-making.	Holistic black-box optimization.
Sample Efficiency	Low. Requiring expensive synthesis tool calls at each step.	High. Using a surrogate model to minimize expensive tool calls.
Reward Handling	Relying on dense rewards.	Designed for sparse rewards.
Branching Factor	Effective for manageable, discrete action spaces.	Struggling with a large branching factor.
Parameterized Operators	Difficulty with hybrid or continuous spaces.	Well-suited for continuous or mixed spaces.
Tool Runtime	Viable if "runtime-per-step" is cheap.	Viable if "runtime-per-evaluation" is expensive.
Generalizability	Classic RL has poor generalizability, while offline RL or pre-training can achieve great generalizability.	None. It is an inherent "per-design" optimizer; the model is not reusable.
Scalability Bottleneck	Limited by GNN difficulty in circuit size.	Limited by search space dimension.
Multi-Objective Handling	Optimizing a single reward or the weighted sum.	Natively designed for multi-objective.

**Note: The discussions in the table refer to representative methods such as DRiLLS (RL) [70] and BOiLS (DSE) [56].*

is expressed as:

$$R_t = \sum_{i=t}^{n} d^{i-t} r_i \tag{19}$$

where r_i is the immediate reward received at step i, n is the terminal step of the episode, and d is the discount factor ($d \in (0, 1]$) that balances the importance of immediate versus future rewards.

DSE Methods. In contrast to the state-dependent, sequential construction paradigm of RL, the DSE methods treat LO as a black-box evaluation problem. Here, the entire synthesis recipe is the atomic unit of optimization, not the individual actions. These search methods operate on a high-level search space $\mathcal{A}^L$, where L is the length of the recipe and $\mathcal{A}$ is the set of possible operations. The similarity of different recipes is calculated by a kernel function for the search strategy to decide the search directions.

The DSE process iteratively proposes a new holistic sequence based on the QoR scores of previously evaluated sequences. The key components are:

- Search Algorithm. The search algorithm includes: (1) Bayesian Optimization (BO) [51, 56], (2) MCTS [33, 122], (3) GA, (4) Random search, (5) Simulated Annealing (SA), and even (6) Neural Networks [39].
- Design Point. The design point is the recipe vector in the feature space, as well as the specific function/algorithm with an equivalent and potentially more efficient variant, such as the NN encoding.
- Evaluation. The evaluation can be performed by LS tools or alternatively, using estimators or ML predictors to produce cost metrics such as QoR.

RL vs. DSE Comparison. While both RL and DSE aim to find an optimal synthesis recipe, their exploration mechanics, computational costs, and application scenarios are fundamentally different. RL typically models the task as an MDP, learning a policy through trial and error to construct a recipe sequentially. DSE treats the entire recipe as a single "black-box" input and attempts to find the global optimum in the fewest evaluations possible by learning a surrogate model. We summarize the direct comparison and ideal scenarios for these two approaches in Table 5.

Representative Works. The earliest RL approach in LS was proposed by [59], which applies the GCN to MIGs to predict the probabilities for each available action (subgraph algorithm) to

apply. DRiLLS [70] introduces the Advantage Actor-Critic (A2C) method on exploration. DRiLLS characterizes the AIG features (e.g., counts of primary I/Os, nodes, edges, levels, and percentages of AND/NOT gates) specifically as the state vector for the A2C agent. The custom AIG feature representation and A2C learning allow DRiLLS to autonomously discover effective optimization recipes. The method in [196] utilizes GCN to extract graph features as state representation, and the REINFORCE agent for RL exploration. The method in [172] extends graph-based RL by adding an LSTM to capture temporal context. The environment uses the AIG structural feature embedded by the GIN and the historical transformation information embedded by the LSTM as the state representation. Then the combined state is fed into the Proximal Policy Optimization (PPO) agent to select the next optimization step. EasySO [182] reframes LS as a hybrid *discrete–continuous* optimization. Rather than only selecting discrete optimizations, its PPO agent simultaneously selects the next operator and tunes its numeric parameters, which covers all LS stages (LO, TM, and post-mapping optimizations). FlowTune [111] adopts the multi-armed bandit algorithm to select actions for both AIG and MIG optimization in different synthesis stages (Boolean minimization, post-mapping, and placement optimization). The method in [125] proposes a unified RL framework that can operate on multiple logic representations (AIG, MIG, LUT networks) to cover different synthesis phases.

In DSE-based methods, BOiLS [56] first introduces BO to navigate the DSE in LO by treating the recipe exploration as a continuous optimization over QoR. The Gaussian process models the cost and guides the selection of operator sequences. The method in [51] combines RL with BO to target multi-objective trade-offs. It uses RL to propose candidate recipes and then applies BO to fine-tune flow hyperparameters, seeking the Pareto front of each action. The method in [39] reformulates the hyperparameter tuning as a classification problem. Rather than sequentially searching, they treat each choice of optimization parameters as a class label. A supervised classifier is trained on features of the circuit to predict the best configuration. By avoiding expensive searches, the classifier can quickly generate recipe parameters. A notable exception is AlphaSyn [122]. AlphaSyn applies a neural network, which is trained through an RL process, to predict both the estimated long-term return and policy priors. These predictions are then used to improve and guide the MCTS policy selection. It proves that core DSE challenges, such as the "long-term effect", can be successfully modeled and resolved using an RL framework.

5.4 Optimization Algorithms Improvement

Another method for optimizing the Boolean circuit is to improve the optimization algorithm with ML methods. Most of the existing methods focus on the rewriting and balancing algorithm. The rewriting operation traverses the graph to find opportunities to replace a k-feasible cut with a logic-equivalent form, given that the replacement brings about the most node count decrease [107].

AISYN [121] modifies the traditional greedy-based approaches in AIG-rewriting with RL and representation learning methods, which model the AIG-rewriting as an MDP. The state is the AIG representation, and the action is the different k-feasible cuts to be selected for rewriting, and demonstrates this problem can be addressed by RL methods, which have the potential to find higher-quality solutions than purely greedy heuristics. The method in [115] takes the selection of the most suitable node-rewriting algorithm as an action in the optimization process and formulates it in an RL framework. The method in [160], named PruneX, proposes a circuit domain generalization framework. It aims to learn domain-invariant representations using GNNs, based on transformation-invariant domain knowledge. Instead of modifying the optimization recipe, PruneX incorporates the learned classifiers within specific optimization algorithms. The classifier then predicts and filters out target nodes for which a transformation would be ineffective, thereby reducing the number of

ineffective node-level applications to accelerate the operator. The methods in [130, 148, 199] utilize RL and DSE to optimize the specific logic designs, which lack generality for common designs.

6 Technology Mapping

TM is the process of transforming a technology-independent Boolean network into a netlist composed of primitive cells from a specific technology library. The initial network is typically represented as a k-bounded subject graph, where each node has at most k fan-ins, such as the AIG and MIG.

The mapping algorithm is typically formulated as a three-step process: (1) Identifying k-feasible cuts using a rapid enumeration procedure [120], (2) matching these cuts to library elements via Boolean matching [14], and (3) selecting a graph cover that optimizes a specific cost function (e.g., area or delay) while adhering to given constraints. This leads to distinct optimization goals, such as delay-driven mapping and area-driven mapping [94]. Table 6 shows how ML models are integrated in TM according to the process.

6.1 Cut Feature Sets

The effectiveness of ML models in TM comes from their ability to handle high-dimensional data, moving beyond simple scalar heuristics.

Structural Features. ML models, particularly GNNs, excel at processing complex graph data. This allows them to directly process the raw graph topology as a high-dimensional feature, surpassing simple scalar values like "volume" or "area flow". The AiMap+ framework [96] utilizes ML to process the local topology by applying feature engineering to transform each cut into a feature set. This set captures the cut's topological signature, boundary properties, and structural information. This high-dimensional representation allows the ML model to distinguish between cuts that are optimal for area versus delay, even if they have identical scalar properties [96].

Reconvergence, a critical non-local feature where paths fan out and later merge, is crucial to capture within a cut to prevent logic duplication [27]. While traditional mappers struggle to capture this structure, ML models can utilize reconvergence in two distinct ways: (1) Predicting and selecting the optimal optimization strategy for a reconvergence cut [115], and (2) Guiding resynthesis on the mapped netlist by identifying reconvergence paths [84].

Timing Features. Estimated slack, which predicts the final timing margin, is a critical component of the mapping algorithm's cost function. It is crucial for guiding the mapper's delay-area trade-off. However, directly predicting slack with ML is challenging due to its global and highly sensitive nature, making it a difficult prediction target.

Consequently, alternative strategies have emerged. One alternative is to predict fundamental components, such as cell and net delays, rather than the final slack value [191]. Another is to use estimated slack as an input feature for an RL agent that guides a related logic optimization process [197].

6.2 Mapper Guidance

Cut Selection. Traditional TM optimization algorithms, such as FlowMap [36] and CutMap [37], focus on cut selection and graph matching. Both use the maximum-flow algorithm to identify "good" cuts for depth reduction, but often at the expense of area and runtime. Priority cuts [109] were introduced to address this, generating only a small, fixed number of "good" k-feasible cuts at each node. This approach achieves high-quality area optimization while minimizing memory and runtime overheads.

SLAP [112] is the first framework to leverage ML for guiding this cut-pruning process. Traditional cut selection in tools like ABC [19] relies on handcrafted heuristics, often sorting cuts by a single

Table 6. ML insertion points in TM

Insertion Point	Objective	Task	ML Technique	Advantage/Problem Solved
Feature Representation	Extracting high-dimensional, non-local features.	Predicting fundamental component delays.	GNN [96, 191]	Embedding the high-dimensional data.
Mapper Guidance	Optimizing the cut-pruning process and reducing runtime overhead.	Multi-class classification	CNN [112], Transformer [174]	Enabling non-local choices and improving prediction accuracy.
Library Tuning	Guiding heuristic mappers by constraining the search space.	Selecting an optimal subset of cells as input to the mapper.	RL [97, 179]	Optimizing the mapper's input and preventing local optima.
Cost Function Modification	Replacing traditional static estimators with high-fidelity predictive models.	Predicting delays of the cut.	GNN [95, 96]	Providing more precise cost estimation, empowering the mapper to select truly optimal cuts.

static attribute, e.g., number of leaves. This is an inherently "local" choice, lacking precise timing and area information at the technology-independent stage, forcing a trade-off between algorithm complexity and QoR. SLAP reformulates cut pruning as a multi-class classification problem. It operates on the hypothesis that the cut's structure itself carries enough information to enable non-local choices. It uses multiple structural features, including node embeddings, leaf levels, fanouts, and cut volume, to construct a cut embedding, then trains a CNN to predict its contribution to the final QoR. Building on SLAP, the method in [174] introduced an attention mechanism to extract more critical information from the cut and improve prediction accuracy.

Library Tuning. "Library tuning" addresses a core TM challenge: providing a massive standard cell library to a mapper can paradoxically degrade the final circuit QoR. This occurs because the vast search space "confuses" the mapper's heuristics, leading to sub-optimal, local choices. The central hypothesis is that a smaller, intelligently selected "partial library", tailored to the design, will constrain the search space and guide the existing heuristic mapper to a superior solution.

Frameworks like MapTune [97] and FuseMap [179] employ RL to find this optimal subset. The "cell selection" problem is modeled as an RL task: the agent "action" is to select a specific cell from the full library to include in the partial library. After a full subset "state" is chosen, the computationally expensive mapping flow is executed. The final circuit QoR serves as the "reward". By iterating this process, often using Multi-Armed Bandit strategies, the agent learns which cell combinations produce the best results.

This method functions as a typical ML-Agent. It does not modify the mapper's internal algorithms (like cut selection). Instead, it optimizes the input given to the mapper. By pruning the library pre-mapping, this method removes poor cell choices that could trap the heuristics, which allows the mapper to operate more effectively within a smaller and higher-quality search space, achieving QoR improvements for both Application-Specific Integrated Circuit (ASIC) [97] and FPGA [179] flows.

Cost Function Modification. Rather than pruning the search space, this application leverages ML to directly replace the inaccurate cost functions that drive the mapper's core optimization algorithm.

Traditional mappers rely on static, statistical heuristics to estimate cost:

- Delay: The primary estimator is the Wire-Load Model (WLM), a statistical look-up table that approximates interconnect resistance and capacitance based on fanout.

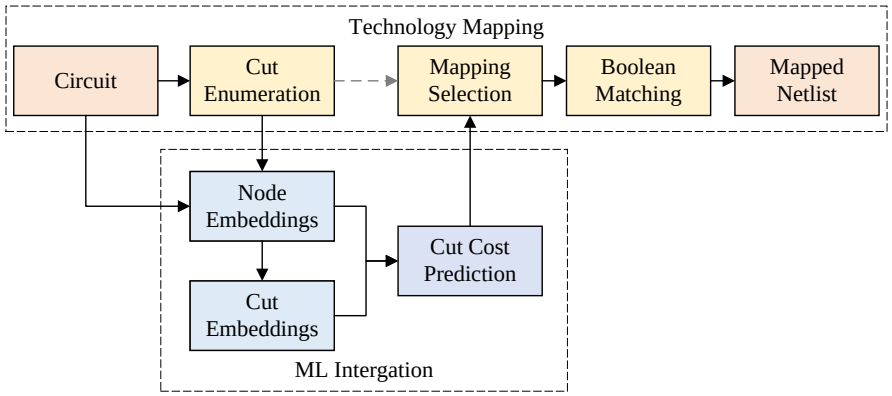

Fig. 8. A general ML framework integrating into technology mapping, which is used to predict the cut cost.

- Area: The estimator cost function is typically a simple summation of static cell areas derived from the technology library [102].

However, these methods are physically agnostic, often selecting cuts that are logically optimal but physically suboptimal due to a lack of layout awareness.

ML improves mapping quality by substituting these static proxies with high-fidelity predictive models. Acting as direct surrogates for internal evaluation functions, these models analyze the high-dimensional features of a candidate cut to generate a precise cost estimate. Fig. 8 illustrates a general framework for this ML integration, where the model is used to predict the cost of various cuts.

- Delay: AiMap [95] utilizes a deep learning framework that integrates an attention mechanism to weight specific pin delays and employs CNNs to fuse structural logic features with standard cell physical data. This allows the mapper to predict load-dependent supergate delays with high fidelity, rejecting cuts that appear logically optimal but suffer from excessive physical latency due to complex connectivity or fanout.
- Area: Research applying ML specifically to area-driven mapping is less prevalent. Historically, area minimization efforts have focused on the FPGA domain, targeting the reduction of LUT counts. A related method [193] applies RL to optimize area costs in optimization. Although distinct from mapping, this methodology offers significant potential for adaptation.

In conclusion, the integration of ML models improves the mapping process by providing more precise cost functions. The enhanced accuracy empowers the mapper to distinguish candidate cuts that are truly optimal from those that are merely locally favorable. Consequently, this paradigm significantly boosts the QoR, achieving netlists with superior timing performance and reduced area compared to traditional mapping heuristics.

7 ML Datasets and Tools

The actual capability of problem-solving from ML is mainly due to advances in the computational capability of current computing devices and the introduction of big data.

7.1 Existing Datasets

There are several initiatives to provide datasets about circuit designs at distinct levels of abstraction and with relevant details for different EDA problems.

- **OpenABC-D** is an extensive labeled dataset created by synthesizing open-source designs with ABC [19, 34]. This dataset offers a valuable resource for developing, evaluating, and benchmarking ML-aided LS techniques. OpenABC-D includes 870,000 AIGs from 1500 synthesis runs, with labels like optimized node counts and delay metrics. Area and delay data are obtained by mapping AIGs using the NanGate 45 nm library.
- **The IWLS contest of 2020** proposes introducing the LO problem as a logic learning problem [126]. The contest has provided an ML dataset composed of 100 incomplete truth tables, each with a single primary output. The goal is to learn a Boolean function from a training set consisting of a small sample of random minterms of the function and try to infer the probable values for the unknown set. The benchmark set also includes image classification instances based on MNIST [44], and CIFAR-10 [86].
- **The IWLS contests of 2022 and 2023** provide a set of 100 benchmarks, including the binary truth tables composed of 0's and 1's. The goal is to synthesize the completely specified multi-output circuits in AIG or XAIG format [2].
- **The CircuitNet 2.0** provides the cross-stage multi-modal data generated by commercial EDA flow and tools with a convenient data format for various ML tasks [23, 76].
- **CircuitGen** is a pseudo-circuit generation framework, and can generate RTL and logic-level circuits on a large scale based on ML prediction, user specifications, and predefined constraints [98].
- **ForgeEDA** is a comprehensive multimodal dataset composed of RTL, AIGs, post-mapping, and placed netlists across diverse design types, such as RISC-V cores and accelerators [143].

7.2 Tools

This section reviews the available LS tools, including the synthesis flow tools and specialized tools designed for ML dataset generation.

- **ABC** is the well-known open-source system for LS based on AIGs. It supports a wide range of optimizations, technology mapping, and equivalence checking, and is highly scriptable for research and benchmarking [19].
- **Yosys** is an open-source synthesis tool, which integrates ABC as a backend for advanced AIG-level optimization and supports RTL-to-netlist generation [164].
- **Mockturtle** is a logic network library, which provides several logic network implementations (such as AIGs, MIGs, and LUTs) and generic algorithms for LS and logic optimization [99].
- **ALSO** is based on the EPFL Libraries, and provides advanced LS tools for both modern FPGA and emerging nanotechnologies [198].
- **CULS** is a GPU-accelerated LS tool and uses GPU parallelism to provide high-performance implementations of common AIG optimization algorithms [3].
- **OpenLS-DGF** introduces an LS dataset generation framework that covers three pivotal stages[114]. By using the framework, OpenLS-DGF can generate an adaptive LS dataset similar to OpenABC-D.
- **OpenLANE** is built out of open-source EDA tools from different projects such as OpenROAD [4] and Qflow [1], providing the overall flow from RTL to layout [139]. In LS, the OpenLANE provides several synthesis strategies that are available for different RTL designs.

7.3 Benchmark Suites

Modern industrial circuit designs often scale to millions of logic gates. Applying GNNs to such designs requires passing messages through enormous computational neighborhoods, a process that is often computationally prohibitive for academic research. To address these practical limitations,

researchers rely on benchmark suites of a manageable scale. These suites facilitate the development of new methods, such as generating task-specific labeled data via LS tools [88] or applying self-supervised learning to unlabeled circuits [166], without the burden of full industrial complexity.

The use of standard benchmarks in microelectronics dates back to ISCAS 1985 [64]. Since then, the landscape has expanded significantly, driven largely by conferences and contests like the IWLS [9], which has introduced numerous suites following classical EDA flows.

Conventional benchmarks usually imitate real circuits, while larger instances are often formed by replicating existing designs or injecting synthetic logic [113]. However, a persistent limitation in EDA datasets is the lack of known optimal implementations. To address this, approaches like those in [113] generate synthetic circuits with known exact solutions. Beyond synthetic logic, the community also utilizes diverse resources such as OpenCores for open-source IP cores [117] and ITC99 [38] for RTL ATPG standards.

8 Discussions

In this section, we discuss the limitations and prospects of ML techniques for LS, extending beyond the entire development process that involves data, algorithms, implementation, and targets.

8.1 Limitations and suggestions

Despite these achievements and the progress of ML applications in LS, there are still several limitations remaining. For each limitation, we not only analyze the root cause but also try to propose potential suggestions.

- **Data Gaps**. Data is fundamental to ML, yet datasets in LS often lack consistency in format and scope. LS data are inherently heterogeneous: some are graph-structured (AIG/MIG), while others consist of coding languages (RTL descriptions), logic literals, or numerical attributes. This heterogeneity limits ML model generalization. As described in Section 5.1, supervised learning methods often struggle to fuse different representations when predicting the QoR of mapped circuits. While some efforts utilize self-supervised learning to align Verilog and AIG representations [166], aligning modalities without clear correspondences remains a significant challenge. Additionally, significant domain gaps exist between varying benchmark suites and industrial designs. Models trained on specific circuit datasets often fail to generalize to unseen datasets due to distributional shifts.
  To bridge these gaps, we propose some strategies focusing on these problems.
  - Unified data generation. Instead of relying on fragmented datasets, the community should adopt unified generation frameworks. Tools like OpenLS-DGF [114] allow for adaptive dataset generation across synthesis stages, while platforms like CircuitNet [23, 76] provide aligned, cross-stage data. These tools provide strict, one-to-one mappings of identical designs across different representation formats.
  - Domain Adaptation. To handle feature disparities and distribution shifts, researchers should leverage cross-domain fusion and domain adaptation techniques. GenEDA [49] advances this field by proposing a Large Language Model (LLM) decoder that achieves functional reasoning through cross-modal alignment between circuit encoders and text embeddings, effectively bridging the gap between different circuit representations.
- **Inconsistent standards**. Current research lacks a unified evaluation framework. For instance, QoR prediction models are evaluated using inconsistent metrics such as MAE [171], MSE [34, 187], MAPE [77, 167, 181, 191], and self-defined reduction factors [12]. Beyond metrics, most studies, across recipe exploration, logic learning, and representation learning, rely on disparate benchmark suites, arbitrary train/test splits, and varying tool versions. This

Table 7. Recommended Metrics and Decision Risk of ML in LS Tasks

Task Domain	Metrics	Explanation	Decision Risk
Boolean Circuit Generation	Accuracy	Functional equivalence	Strict verification required, but possible security removal
	#Nodes & #Levels	Proxy for QoR	Physical unfeasibility
QoR Prediction	MAE, MSE, MAPE	Prediction fidelity per design	Misleading recipe ranking
	Ranking (e.g., NDCG@k)	Recipe ranking accuracy	Blindness to absolute constraints and time waste
Logic Optimization	Top-1 Improvement	Peak QoR gain vs. baseline	Excessive search cost
	Search Trials & Time	Convergence speed	May be trapped in local optima
Technology Mapping	End-to-End QoR	Post-TM result or Post-P&R result	Inference latency bottleneck
	Cut Ranking	Identification of good cuts	Local-global objective mismatch

inconsistency makes cross-paper comparisons nearly impossible and unfair. Such disparity leads to (1) overfitting risks on small or proprietary datasets and (2) misleading performance claims, where a metric value (e.g., 1% MAPE) implies accuracy in one context but failure in another.

To address this, we identify the mapping between statistical metrics and industrial decision risks. As detailed in Table 7, relying on inappropriate metrics can mask critical engineering failures, such as timing closure violations or computational waste. In our opinion, the community requires a reference evaluation protocol and centralized benchmarking platforms.

- Unified metric reporting. Researchers are strongly encouraged to incorporate the full set of recommended metrics listed in Table 7. A holistic evaluation that covers statistical fidelity, ranking capability, and engineering constraints is essential to prevent biased conclusions.
- Open-source leaderboards. Similar to evaluation standards in Computer Vision (e.g., ImageNet [43]), the field should establish an open-source online evaluation platform. This platform would host containerized environments (e.g., Docker) with fixed logic synthesis tools and standard datasets. By submitting models to a unified test harness, we can ensure fair comparison, reproducibility, and clear identification of state-of-the-art performance.

- **Interpretability**. Like other ML models, GNNs are complex "black boxes" for LS. They lack the interpretability to explain why a specific optimization decision (e.g., a cut selection in TM) is superior. This lack of transparency creates a trust barrier: even if an ML-generated circuit passes logical equivalence checking, engineers may hesitate to adopt it if the structure appears counter-intuitive or poses potential risks for physical design. Merely comparing results with traditional algorithms proves capability but fails to provide insight.

  To increase trust and human-ML collaboration, we propose some strategies inspired by both ML and LS.

  - Explainable architecture. Researchers can employ more explainable ML tools specifically for graphs (e.g., GNNExplainer [176]). These tools can identify and visualize the critical subgraphs or feature attributions that contributed most to a model's prediction, converting intransparent numerical weights into structural logic insights, and in turn deepening understanding of Boolean circuits.

  - Prioritizing ML-Assistance. We recommend favoring the ML-Assistance paradigm over the ML-Agent approach. The ML model does not necessarily directly replace the synthesis tool; instead, it acts as a guide by predicting heuristics or search directions. While the actual synthesis operations are executed by trusted, deterministic symbolic engines (e.g., ABC [19]), this ensures the process remains interpretable and verifiable, as the final circuit is constructed by well-understood algorithms.
- **Limited tasks**. Most existing works focus on combinational and gate-level circuits. Sequential circuits, multi-bit arithmetic units, asynchronous circuits, and complex physical synthesis tasks are largely untouched. Focusing solely on combinational logic often leads to local optima, as improvements in logic depth or size may disappear after Place and Route (P&R). Furthermore, TM remains mostly heuristic-driven; existing ML works in TM primarily support cut selection rather than directly generating mapped netlists. Extending ML techniques to these broader and downstream-aware areas remains an open challenge.
  To expand the scope of ML in LS, several works have been proposed to address the problem:
  - Generalized circuit modeling. ML architectures must evolve to capture the unique characteristics of diverse circuit types beyond simple combinational logic. For instance, in the case of sequential circuits, models should transition from static DAGs to Sequential GNNs. This involves treating FFs as special temporal nodes or learning state transition features (similar to approaches in [80, 81]), thereby enabling message-passing across clock boundaries. Similarly, specific inductive biases should be introduced for arithmetic or asynchronous structures.
  - LLM-driven task generalization: To significantly broaden the problem-solving scope, researchers can leverage LLMs. Unlike task-specific GNNs, LLMs can process multi-modal inputs (e.g., RTL code, synthesis logs, and constraints) to handle diverse tasks that traditional models struggle with. However, LLMs also face problems such as difficulty in graph learning and massive resource consumption, posing challenges for deployment in cost-sensitive EDA flows [48, 119].

8.2 Prospects

Despite these limitations and suggestions discussed in Section 8.1, there are some potential directions for researchers to explore.

- **New ML schemes**. Classical ML-based methods typically adopt a bottom-up approach, simulating specific processes or predicting isolated results (e.g., RL for optimization). The future may lie in redesigning the synthesis flow to be **ML-Native**. While section 8.1 addressed the technical need for hybrid and LLM-based solutions, the broader prospect is how these schemes fundamentally alter the design ecosystem. The circuit generation process discussed in Section 3 may reflect the ML-Native scheme, if there is no additional fine-tuning. We anticipate a move towards autonomous design agents that can self-correct, negotiate trade-offs without human intervention, and generate the target circuit autonomously. The new ML-Native scheme not only inspires exploration of non-intuitive circuit structures that traditional human logic has never imagined, but also lowers entry barriers by allowing non-experts to use natural language for high-quality synthesis, thus potentially reshaping the semiconductor industry and business landscape.
- **Safety**. The integration of ML into LS introduces a dual-front challenge regarding safety and security. On one hand, the ML models themselves (e.g., QoR predictors) are vulnerable to adversarial attacks, where slight circuit adjustments could cause wildly different predictions, misleading the synthesis flow. On the other hand, ML techniques have proven alarmingly

effective at invalidating existing IP protection strategies, as discussed in Section 4.4. The potential future direction is not only to advance ML algorithms to make ML difficult to attack, but also to redesign hardware to have inherent structures and features that are difficult to decipher by ML, making it difficult to launch attacks.

- **Scalability**. The increasing scale of circuits poses severe scalability challenges to GNNs. Developing a GNN that satisfies the industrial scale is computationally impractical due to the neighborhood explosion phenomenon [183]. While segmenting large circuits into partitions is an efficient data-side method (as exemplified in Section 5.2 and [192]), this approach is often unexplored in broader contexts and risks losing global context information. The future direction may lie in hierarchical graph learning [177]. Besides, applying ML to simplify large circuits is also a feasible method. BoolSkeleton [116] proposes a Boolean network skeletonization technique via homogeneous pattern reduction, which simplifies the graph structure while preserving essential functional information. Future models may adopt multi-level pooling techniques or transfer learning (training small circuits to generalize to large circuits), rather than directly processing large circuits.
- **Implementations**. To enable practical industrial adoption, improvements must be made from both the model and flow perspectives. Currently, high-accuracy models are often computationally prohibitive for logic synthesis, while lightweight predictive models may lack robustness. The possible direction is applying network pruning, quantization, and knowledge distillation to create "Tiny" versions of GNNs that can run within millisecond latency budgets inside C++ synthesis kernels. Furthermore, future ML model implementations will be moving beyond static offline training to online and continuous learning, allowing tools to learn as they synthesize, and becoming smarter with every design iteration performed by the customer.

9 Conclusion

The rapid progress of ML in LS is motivated by sophisticated synthesis flows and frameworks, which provide a multitude of tasks for ML to explore, as well as the EDA infrastructures that are capable of processing and generating circuit data, founding the basis of ML models. The survey broadly shows the ML applications in different LS flows, which can be categorized into two patterns: (1) the ML-Agent method that directly leverages ML as a synthesis tool, and (2) the ML-Assistance method that involves predicting metrics or some intermediate variables or criteria of interest. The methods of ML in LS are not limited to GNN and its variants, although circuits and netlists can be naturally represented as graphs. The RL and DSE methods are also widely utilized in LS, both in improving the efficiency of LS algorithms and in the exploration of optimization recipes. Moreover, the image-based CNNs, text-based RNNs and Transformers are also used for specific tasks. We review the emerging ML datasets and tools developed specifically for ML tasks, reflecting the increasing maturity of the field. We also discuss the limitations, potential suggestions, and future prospects of ML applications in LS and industrial production, hoping the ML techniques can be beneficial and revolutionary in LS.

References

[1] 2012. Open Circuit Design. http://opencircuitdesign.com.
[2] 2022. International Workshop on Logic and Synthesis Contest. https://www.iwls.org/contest/
[3] Evangeline F.Y. Young et al. 2022. CULS. https://github.com/cuhk-eda/CULS.
[4] Tutu Ajayi and David Blaauw. 2019. OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference*.
[5] Lilas Alrahis, Satwik Patnaik, et al. 2021. UNTANGLE: Unlocking Routing and Logic Obfuscation Using Graph Neural Networks-based Link Prediction. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

doi:10.1109/ICCAD51958.2021.9643476

[6] Lilas Alrahis, Satwik Patnaik, et al. 2022. MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural Network-based Link Prediction. In *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 694–699. doi:10.23919/DATE54114.2022.9774603

[7] Lilas Alrahis, Satwik Patnaik, et al. 2022. OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 3 (2022), 1602–1606. doi:10.1109/TCSII.2021.3113035

[8] Lilas Alrahis, Abhrajit Sengupta, et al. 2022. GNN-RE: Graph Neural Networks for Reverse Engineering of Gate-Level Netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 8 (2022), 2435–2448. doi:10.1109/TCAD.2021.3110807

[9] Luca Amarú, Pierre-Emmanuel Gaillardon, et al. 2015. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*.

[10] Luca Amarú, Pierre-Emmanuel Gaillardon, et al. 2014. Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization. In *Proceedings of the 2014 51st ACM/EDAC/IEEE Design Automation Conference*. IEEE, 1–6.

[11] Saeed Amizadeh, Sergiy Matusevych, et al. 2019. Learning To Solve Circuit-SAT: An Unsupervised Differentiable Approach. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJxgz2R9t7

[12] Animesh Basak Chowdhury, Benjamin Tan, et al. 2023. Bulls-Eye: Active Few-Shot Learning Guided Logic Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 8 (2023), 2580–2590. doi:10.1109/TCAD.2022.3226668

[13] Peter Belcak and Roger Wattenhofer. 2022. Neural Combinatorial Logic Circuit Synthesis from Input-Output Examples. arXiv:2210.16606 [cs.LG] https://arxiv.org/abs/2210.16606

[14] Luca Benini and Giovanni De Micheli. 1997. A survey of Boolean matching techniques for library binding. 2, 3 (July 1997), 193–226. doi:10.1145/264995.264996

[15] Augusto Berndt and Cristina Meinhardt. 2024. CeConP: Exploring Node Centrality for Early Routing Congestion Prediction. In *2024 IEEE 15th Latin America Symposium on Circuits and Systems (LASCAS)*. 1–5. doi:10.1109/LASCAS60203.2024.10506148

[16] Augusto André Souza Berndt, Mateus Fogaça, et al. 2022. A Review of Machine Learning in Logic Synthesis. *Journal of Integrated Circuits and Systems* 17, 3 (2022), 1–12. doi:10.29292/jics.v17i3.649

[17] George EP Box, Gwilym M Jenkins, et al. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[18] Robert Brayton and Alan Mishchenko. 2006. Scalable Logic Synthesis using a Simple Circuit Structure. In *Proceedings of the International Workshop on Logic Synthesis (IWLS)*, Vol. 6.

[19] Robert Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification (Lecture Notes in Computer Science, Vol. 6174)*. Springer, 24–40. doi:10.1007/978-3-642-14295-6_5

[20] Benedikt Bünz and Matthew Lamm. 2017. Graph Neural Networks and Boolean Satisfiability. arXiv:1702.03592 [cs.AI] https://arxiv.org/abs/1702.03592

[21] Ye Cai, Rui Wang, et al. 2025. AiLO: A Predictive Framework for Logic Optimization Using Multi-Scale Cross-Attention Transformer. *ACM Transactions on Design Automation of Electronic Systems* (2025).

[22] Chris Cameron, Rex Chen, et al. 2020. Predicting Propositional Satisfiability via End-to-End Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3324–3331. doi:10.1609/aaai.v34i04.5733

[23] Zhuomin Chai, Yuxiang Zhao, et al. 2023. CircuitNet: An Open-Source Dataset for Machine Learning in VLSI CAD Applications With Improved Domain-Specific Evaluation Metric and Learning Strategies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 12 (2023), 5034–5047. doi:10.1109/TCAD.2023.3287970

[24] Prabuddha Chakraborty, Jonathan Cruz, et al. 2018. SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation. In *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 56–61. doi:10.1109/AsianHOST.2018.8607163

[25] Wenjing Chang and Wenlong Liu. 2025. SAT-GATv2: A Dynamic Attention-Based Graph Neural Network for Solving Boolean Satisfiability Problem. *Electronics* 14, 3 (2025). doi:10.3390/electronics14030423

[26] Wenjing Chang, Hengkai Zhang, et al. 2022. Predicting Propositional Satisfiability Based on Graph Attention Networks. *International Journal of Computational Intelligence Systems* 15 (2022), 84. doi:10.1007/s44196-022-00139-9

[27] Deming Chen and Jason Cong. 2004. DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 752–759.

[28] Fenxiao Chen, Yun-Cheng Wang, et al. 2020. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing* 9 (2020), e15. doi:10.1017/ATSIP.2020.13

[29] Huili Chen, Cheng Fu, et al. 2022. GALU: A Genetic Algorithm Framework for Logic Unlocking. *Digital Threats* 4, 2, Article 21 (Feb. 2022), 30 pages. doi:10.1145/3491256

[30] Ziliang Chen and Zhanfu Yang. 2019. Graph Neural Reasoning May Fail in Certifying Boolean Unsatisfiability. arXiv:1909.11588 [cs.LG] https://arxiv.org/abs/1909.11588

[31] Mason Chern, Shih-Wei Lee, et al. 2019. Improving scan chain diagnostic accuracy using multi-stage artificial neural networks. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '19)*. Association for Computing Machinery, New York, NY, USA, 341–346. doi:10.1145/3287624.3287692

[32] Animesh B. Chowdhury, Jitendra Bhandari, et al. 2023. ConVERTS: Contrastively Learning Structurally InVariant Netlist Representations. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. 1–6. doi:10.1109/MLCAD58807.2023.10299862

[33] Animesh Basak Chowdhury, Marco Romanelli, et al. 2024. Retrieval-Guided Reinforcement Learning for Boolean Circuit Minimization. arXiv:2401.12205 [cs.LG] https://arxiv.org/abs/2401.12205

[34] Animesh Basak Chowdhury, Benjamin Tan, et al. 2021. OpenABC-D: A Large-Scale Dataset For Machine Learning Guided Integrated Circuit Synthesis. *CoRR* abs/2110.11292 (2021). arXiv:2110.11292 https://arxiv.org/abs/2110.11292

[35] Zhufei Chu, Mathias Soeken, et al. 2019. Structural rewriting in XOR-majority graphs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 663–668.

[36] J. Cong and Yuzheng Ding. 1994. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13, 1 (1994), 1–12. doi:10.1109/43.273754

[37] Jason Cong and Yean-Yow Hwang. 1995. Simultaneous depth and area minimization in LUT-based FPGA mapping. In *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '95)*. Association for Computing Machinery, New York, NY, USA, 68–74. doi:10.1145/201310.201322

[38] F. Corno, M.S. Reorda, et al. 2000. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design and Test of Computers* 17, 3 (2000), 44–53. doi:10.1109/54.867894

[39] Zhenghao Cui and Minghua Shen. 2024. Automatic Multi-Parameter Tuning for Logic Synthesis with Reinforcement Learning. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. 318–323. doi:10.1109/ISEDA62518.2024.10617552

[40] Márk Danisovszky, Zijian Gyozo Yang, et al. 2020. Classification of SAT Problem Instances by Machine Learning Methods.. In *ICAI*. 94–104.

[41] Armin Darjani, Nima Kavand, et al. 2024. Flip-Lock: A Flip-Flop-Based Logic Locking Technique for Thwarting ML-based and Algorithmic Structural Attacks. In *Proceedings of the Great Lakes Symposium on VLSI 2024* (Clearwater, FL, USA) *(GLSVLSI '24)*. Association for Computing Machinery, New York, NY, USA, 185–191. doi:10.1145/3649476.3658753

[42] Stéphane d'Ascoli, Samy Bengio, et al. 2023. Boolformer: Symbolic Regression of Logic Functions with Transformers. arXiv:2309.12207 [cs.LG] https://arxiv.org/abs/2309.12207

[43] Jia Deng, Wei Dong, et al. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. doi:10.1109/CVPR.2009.5206848

[44] L. Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine* 29, 6 (Nov 2012), 141–142.

[45] David Devlin and Barry O'Sullivan. 2008. Satisfiability as a classification problem. In *Proc. of the 19th Irish Conf. on Artificial Intelligence and Cognitive Science*.

[46] Faezeh Faez, Raika Karimi, et al. 2025. MTLSO: A Multi-Task Learning Approach for Logic Synthesis Optimization. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '25)*. Association for Computing Machinery, New York, NY, USA, 72–78. doi:10.1145/3658617.3697721

[47] C. Fagot, P. Girard, et al. 1997. On using machine learning for logic BIST. In *Proceedings International Test Conference 1997*. 338–346. doi:10.1109/TEST.1997.639635

[48] Wenji Fang, Jing Wang, et al. 2025. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA. arXiv:2504.03711 [cs.AR] https://arxiv.org/abs/2504.03711

[49] Wenji Fang, Jing Wang, et al. 2025. GenEDA: Towards Generative Netlist Functional Reasoning via Cross-Modal Circuit Encoder-Decoder Alignment. arXiv:2504.09485 [cs.LG] https://arxiv.org/abs/2504.09485

[50] Arash Fayyazi, Soheil Shababi, et al. 2019. Deep Learning-Based Circuit Recognition Using Sparse Mapping and Level-Dependent Decaying Sum Circuit Representations. In *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 638–641. doi:10.23919/DATE.2019.8715251

[51] Chang Feng, Wenlong Lyu, et al. 2022. Batch Sequential Black-Box Optimization with Embedding Alignment Cells for Logic Synthesis. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (San Diego, California) *(ICCAD '22)*. Association for Computing Machinery, New York, NY, USA, Article 56, 9 pages. doi:10.1145/3508352.3549363

[52] Hao Geng, Yuzhe Ma, et al. 2022. High-Speed Adder Design Space Exploration via Graph Neural Processes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 8 (2022), 2657–2670. doi:10.1109/TCAD.2021.3114262

[53] Justin Gilmer, Samuel S. Schoenholz, et al. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*. PMLR,

1263–1272. https://proceedings.mlr.press/v70/gilmer17a.html
[54] Ian Goodfellow, Yoshua Bengio, et al. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[55] Google DeepMind. 2023. Optimizing Computer Systems with More Generalized AI Tools. https://deepmind.google/impact/optimizing-computer-systems-with-more-generalized-ai-tools/
[56] Antoine Grosnit, Cedric Malherbe, et al. 2022. BOiLS: Bayesian Optimisation for Logic Synthesis. In *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 1193–1196. doi:10.23919/DATE54114.2022.9774632
[57] Wenxuan Guo, Hui-Ling Zhen, et al. 2023. Machine Learning Methods in Solving the Boolean Satisfiability Problem. *Machine Intelligence Research* 20, 5 (2023), 640–655. doi:10.1007/s11633-022-1396-2
[58] Kevin Gurney. 2018. *An introduction to neural networks*. CRC press.
[59] Winston Haaswijk, Edo Collins, et al. 2018. Deep Learning for Logic Optimization Algorithms. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. doi:10.1109/ISCAS.2018.8351885
[60] Gary D Hachtel and Fabio Somenzi. 2005. *Logic synthesis and verification algorithms*. Springer Science & Business Media.
[61] Will Hamilton, Zhitao Ying, et al. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf
[62] Jesse Michael Han. 2020. Enhancing SAT solvers with glue variable predictions. arXiv:2007.02559 [cs.LO] https://arxiv.org/abs/2007.02559
[63] Jesse Michael Han. 2020. Learning cubing heuristics for SAT from DRAT proofs. In *Conference on Artificial Intelligence and Theorem Proving (AITP)*.
[64] M. C. Hansen, H. Yalcin, et al. 1999. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design Test of Computers* 16, 3 (Jul 1999), 72–80. doi:10.1109/54.785838
[65] Soha Hassoun and Tsutomu Sasao. 2001. *Logic synthesis and verification*. Vol. 654. Springer Science & Business Media.
[66] Zhuolun He, Ziyi Wang, et al. 2021. Graph Learning-Based Arithmetic Block Identification. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–8. doi:10.1109/ICCAD51958.2021.9643581
[67] Lisa Hellerstein and Rocco A. Servedio. 2007. On PAC Learning Algorithms for Rich Boolean Function Classes. *Theoretical Computer Science* 384, 1 (2007), 66–76. doi:10.1016/j.tcs.2007.05.014
[68] Sean B Holden and others. 2021. Machine learning for automated theorem proving: Learning to solve SAT and QSAT. *Foundations and Trends® in Machine Learning* 14, 6 (2021), 807–989.
[69] Xuenong Hong, Yee-Yang Tee, et al. 2024. GNNReveal: A Novel Graph Neural Network-based Attack Method for IC Logic Gate De-Camouflaging. *IEEE Intelligent Systems* (2024), 1–9. doi:10.1109/MIS.2024.3430263
[70] Abdelrahman Hosny, Soheil Hashemi, et al. 2020. DRiLLS: Deep Reinforcement Learning for Logic Synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 581–586. doi:10.1109/ASP-DAC47756.2020.9045559
[71] Hailiang Hu, Jiang Hu, et al. 2023. Machine-Learning Based Delay Prediction for FPGA Technology Mapping. In *Proceedings of the 24th ACM/IEEE Workshop on System Level Interconnect Pathfinding* (San Diego, California) *(SLIP '22)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. doi:10.1145/3557988.3569713
[72] Yu Huang, Brady Benware, et al. 2017. Scan Chain Diagnosis Based on Unsupervised Machine Learning. In *2017 IEEE 26th Asian Test Symposium (ATS)*. 225–230. doi:10.1109/ATS.2017.50
[73] Yu Huang, Ruifeng Guo, et al. 2008. Survey of Scan Chain Diagnosis. *IEEE Design and Test of Computers* 25, 3 (2008), 240–248. doi:10.1109/MDT.2008.83
[74] Joshua Immanuel and Spencer K. Millican. 2020. Calculating Signal Controllability using Neural Networks: Improvements to Testability Analysis and Test Point Insertion. In *2020 IEEE 29th North Atlantic Test Workshop (NATW)*. 1–6. doi:10.1109/NATW49237.2020.9153082
[75] Eric Jang, Shixiang Gu, et al. 2016. Categorical Reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144* (2016). https://arxiv.org/abs/1611.01144
[76] Xun Jiang, zhuomin chai, et al. 2024. CircuitNet 2.0: An Advanced Dataset for Promoting Machine Learning Innovations in Realistic Chip Design Environment. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=nMFSUjxMIl
[77] Raika Karimi, Faezeh Faez, et al. 2025. Logic Synthesis Optimization with Predictive Self-Supervision via Causal Transformers. arXiv:2409.10653 [cs.AI] https://arxiv.org/abs/2409.10653
[78] Maurice Karnaugh. 1953. The Map Method for Synthesis of Combinational Logic Circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* 72 (1953), 593–599. doi:10.1109/TCE.1953.6371932
[79] Guolin Ke, Qi Meng, et al. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[80] Sadaf Khan, Zhengyuan Shi, et al. 2024. DeepSeq: Deep Sequential Circuit Learning. In *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 1–2. doi:10.23919/DATE58400.2024.10546639
[81] Sadaf Khan, Zhengyuan Shi, et al. 2025. DeepSeq2: Enhanced Sequential Circuit Learning with Disentangled Representations. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '25)*. Association for Computing Machinery, New York, NY, USA, 498–504. doi:10.1145/3658617.3697594
[82] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG] https://arxiv.org/abs/1609.02907
[83] Robert Kirby, Saad Godil, et al. 2019. CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. 217–222. doi:10.1109/VLSI-SoC.2019.8920342
[84] Jitka Kocnová and Zdenek Vasicek. 2021. Resynthesis of logic circuits using machine learning and reconvergent paths. In *2021 24th Euromicro Conference on Digital System Design (DSD)*. 69–76. doi:10.1109/DSD53832.2021.00020
[85] Kun Kong, Yun Shang, et al. 2009. An optimized majority logic synthesis methodology for quantum-dot cellular automata. *IEEE Transactions on Nanotechnology* 9, 2 (2009), 170–183.
[86] A. Krizhevsky, G. Hinton, and others. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto.
[87] Vitaly Kurin, Saad Godil, et al. 2020. Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver?. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 9608–9621. https://proceedings.neurips.cc/paper_files/paper/2020/file/6d70cb65d15211726dcce4c0e971e21c-Paper.pdf
[88] Min Li, Sadaf Khan, et al. 2022. DeepGate: learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 667–672. doi:10.1145/3489517.3530497
[89] Min Li, Zhengyuan Shi, et al. 2023. DeepSAT: An EDA-Driven Learning Framework for SAT. arXiv:2205.13745 [cs.AI] https://arxiv.org/abs/2205.13745
[90] Mufei Li, Viraj Shitole, et al. 2024. LayerDAG: A Layerwise Autoregressive Diffusion Model of Directed Acyclic Graphs for System. In *Machine Learning for Computer Architecture and Systems 2024*. https://openreview.net/forum?id=IsarrieeQA
[91] Xihan Li, Xing Li, et al. 2024. Logic Synthesis with Generative Deep Neural Networks. arXiv:2406.04699 [cs.LO] https://arxiv.org/abs/2406.04699
[92] Xihan Li, Xing Li, et al. 2025. Circuit Transformer: A Transformer That Preserves Logical Equivalence. arXiv:2403.13838 [cs.LG] https://arxiv.org/abs/2403.13838
[93] Hanxiao Liu, Karen Simonyan, et al. 2019. DARTS: Differentiable Architecture Search. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1806.09055
[94] Junfeng Liu, Liwei Ni, et al. 2024. A Delay-Driven Iterative Technology Mapping Framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024), 1–1. doi:10.1109/TCAD.2024.3524463
[95] Junfeng Liu, Liwei Ni, et al. 2023. AiMap: Learning to Improve Technology Mapping for ASICs via Delay Prediction. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. 344–347. doi:10.1109/ICCD58817.2023.00059
[96] Junfeng Liu and Qinghua Zhao. 2024. AiMap+: Guiding Technology Mapping for ASICs via Learning Delay Prediction. *Electronics* 13, 18 (2024), 3614.
[97] Mingju Liu, Daniel Robinson, et al. 2025. MapTune: Advancing ASIC Technology Mapping via Reinforcement Learning Guided Library Tuning. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design* (Newark Liberty International Airport Marriott, New York, NY, USA) *(ICCAD '24)*. Association for Computing Machinery, New York, NY, USA, Article 143, 10 pages. doi:10.1145/3676536.3676762
[98] Shang Liu, Wenji Fang, et al. 2025. *Towards Big Data in AI for EDA Research: Generation of New Pseudo Circuits at RTL Stage*. Association for Computing Machinery, New York, NY, USA, 527–533. https://doi.org/10.1145/3658617.3697613
[99] lsils. 2018. mockturtle: C++ Logic Network Library. https://github.com/lsils/mockturtle.
[100] Yuzhe Ma, Haoxing Ren, et al. 2019. High Performance Graph Convolutional Networks with Applications in Testability Analysis. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 18, 6 pages. doi:10.1145/3316781.3317838
[101] Lakshmi Likhitha Mankali, Ozgur Sinanoglu, et al. 2024. INSIGHT: Attacking Industry-Adopted Learning Resilient Logic Locking Techniques Using Explainable Graph Neural Network. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 91–108. https://www.usenix.org/conference/usenixsecurity24/presentation/mankali
[102] V. Manohararajah, S.D. Brown, et al. 2006. Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 11 (2006), 2331–2340. doi:10.1109/TCAD.2006.882119

[103] Joao P Marques-Silva and Karem A Sakallah. 2002. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on computers* 48, 5 (2002), 506–521.
[104] Giovanni De Micheli. 1994. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York.
[105] Spencer Millican, Yang Sun, et al. 2019. Applying Neural Networks to Delay Fault Testing: Test Point Insertion and Random Circuit Training. In *2019 IEEE 28th Asian Test Symposium (ATS)*. 13–135. doi:10.1109/ATS47505.2019.000-7
[106] Alan Mishchenko, Robert Brayton, et al. 2011. Delay Optimization Using SOP Balancing. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. IEEE, 375–382. doi:10.1109/ICCAD.2011.6105357
[107] Alan Mishchenko, Satrajit Chatterjee, et al. 2006. DAG-aware AIG rewriting. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference*. ACM, 532–535. doi:10.1145/1146909.1147048
[108] Alan Mishchenko, Satrajit Chatterjee, et al. 2005. *FRAIGs: A Unifying Representation for Logic Synthesis and Verification*. Technical Report UCB/ERL M05/10. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. https://people.eecs.berkeley.edu/~alanmi/publications/2005/tech05_fraigs.pdf
[109] Alan Mishchenko, Sungmin Cho, et al. 2007. Combinational and sequential mapping with priority cuts. In *2007 IEEE/ACM International Conference on Computer-Aided Design*. 354–361. doi:10.1109/ICCAD.2007.4397290
[110] Walter Lau Neto, Max Austin, et al. 2019. LSOracle: a Logic Synthesis Framework Driven by Artificial Intelligence: Invited Paper. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–6. doi:10.1109/ICCAD45719.2019.8942145
[111] Walter Lau Neto, Yingjie Li, et al. 2023. FlowTune: End-to-End Automatic Logic Optimization Exploration via Domain-Specific Multiarmed Bandit. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 6 (2023), 1912–1925. doi:10.1109/TCAD.2022.3213611
[112] Walter Lau Neto, Matheus T. Moreira, et al. 2021. SLAP: A Supervised Learning Approach for Priority Cuts Technology Mapping. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 859–864. doi:10.1109/DAC18074.2021.9586230
[113] Walter Lau Neto, Vinicius N. Possani, et al. 2020. Exact Benchmark Circuits for Logic Synthesis. *IEEE Design and Test* 37, 3 (2020), 51–58. doi:10.1109/MDAT.2019.2952348
[114] Liwei Ni, Rui Wang, et al. 2024. OpenLS-DGF: An Adaptive Open-Source Dataset Generation Framework for Machine Learning Tasks in Logic Synthesis. arXiv:2411.09422 [cs.AI] https://arxiv.org/abs/2411.09422
[115] Liwei Ni, Zonglin Yang, et al. 2023. Adaptive Reconvergence-driven AIG Rewriting via Strategy Learning. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. 336–343. doi:10.1109/ICCD58817.2023.00058
[116] Liwei Ni, Jiaxi Zhang, et al. 2025. BoolSkeleton: Boolean Network Skeletonization via Homogeneous Pattern Reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025), 1–1. doi:10.1109/TCAD.2025.3628790
[117] OpenCores Community. 2022. Opencores. https://opencores.org/
[118] Emils Ozolins, Karlis Freivalds, et al. 2022. Goal-Aware Neural SAT Solver. In *2022 International Joint Conference on Neural Networks (IJCNN)*. 1–8. doi:10.1109/IJCNN55064.2022.9892733
[119] Jingyu Pan, Guanglei Zhou, et al. 2025. A Survey of Research in Large Language Models for Electronic Design Automation. arXiv:2501.09655 [cs.LG] https://arxiv.org/abs/2501.09655
[120] Peichen Pan and Chih-Chang Lin. 1998. A New Retiming-Based Technology Mapping Algorithm for LUT-Based FPGAs. In *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*. 35–42. doi:10.1145/275107.275118
[121] Ghasem Pasandi, Sreedhar Pratty, et al. 2023. AISYN: AI-driven Reinforcement Learning-Based Logic Synthesis Framework. arXiv:2302.06415 [cs.AI] https://arxiv.org/abs/2302.06415
[122] Zehua Pei, Fangzhou Liu, et al. 2023. AlphaSyn: Logic Synthesis Optimization with Efficient Monte Carlo Tree Search. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323856
[123] Felix Petersen, Christian Borgelt, et al. 2022. Deep Differentiable Logic Gate Networks. In *Advances in Neural Information Processing Systems*, Vol. 35. 2006–2018. https://proceedings.neurips.cc/paper_files/paper/2022/file/0d3496dd0cec77a999c98d35003203ca-Paper-Conference.pdf
[124] Yuan Pu and others. 2025. HeLO: A Heterogeneous Logic Optimization Framework by Hierarchical Clustering and Graph Learning. In *Proceedings of the International Symposium on Physical Design (ISPD'25)*.
[125] Yu Qian, Xuegong Zhou, et al. 2024. An Efficient Reinforcement Learning Based Framework for Exploring Logic Synthesis. *ACM Trans. Des. Autom. Electron. Syst.* 29, 2, Article 25 (Jan. 2024), 33 pages. doi:10.1145/3632174
[126] Shubham Rai, Walter Lau Neto, et al. 2021. Logic Synthesis Meets Machine Learning: Trading Exactness for Generalization. In *2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 1026–1031. doi:10.23919/DATE51398.2021.9473972
[127] Haoxing Ren and Jiang Hu. 2022. *Machine learning applications in electronic design automation*. Springer.
[128] Lev Reyzin. 2020. Statistical Queries and Statistical Algorithms: Foundations and Applications. *arXiv preprint arXiv:2004.00557* (2020). https://arxiv.org/abs/2004.00557

[129] Christian Ritz and Jens Carl Streibig. 2008. *Nonlinear Regression with R.* Springer Science and Business Media. doi:10.1007/978-0-387-09616-2
[130] Rajarshi Roy, Jonathan Raiman, et al. 2021. PrefixRL: Optimization of Parallel Prefix Circuits using Deep Reinforcement Learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 853–858. doi:10.1109/DAC18074.2021.9586094
[131] Soham Roy. 2021. *Toward Zero Backtracks in Test Pattern Search Algorithms with Machine Learning.* Ph.D. Dissertation. Auburn University. ProQuest Dissertations & Theses, Accession No. 29288865.
[132] Soham Roy, Spencer K. Millican, et al. 2020. Machine Intelligence for Efficient Test Pattern Generation. In *2020 IEEE International Test Conference (ITC)*. 1–5. doi:10.1109/ITC44778.2020.9325250
[133] Soham Roy, Spencer K. Millican, et al. 2021. Training Neural Network for Machine Intelligence in Automatic Test Pattern Generator. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*. 316–321. doi:10.1109/VLSID51830.2021.00059
[134] Soham Roy, Spencer K. Millican, et al. 2024. A Survey and Recent Advances: Machine Intelligence in Electronic Testing. *Journal of Electronic Testing* 40 (2024), 139–158. doi:10.1007/s10836-024-06117-7
[135] Bernhard Schölkopf and Alexander J. Smola. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press. https://mitpress.mit.edu/9780262194754/learning-with-kernels/
[136] George A. F. Seber and Alan J. Lee. 2003. *Linear Regression Analysis* (2 ed.). John Wiley and Sons, Hoboken, NJ. doi:10.1002/9780471722199
[137] Daniel Selsam and Nikolaj Bjørner. 2019. Guiding High-Performance SAT Solvers with Unsat-Core Predictions. In *Theory and Applications of Satisfiability Testing – SAT 2019*. Springer International Publishing, Cham, 336–353.
[138] Daniel Selsam, Matthew Lamm, et al. 2019. Learning a SAT Solver from Single-Bit Supervision. arXiv:1802.03685 [cs.AI] https://arxiv.org/abs/1802.03685
[139] Mohamed Shalan and Tim Edwards. 2020. Building OpenLANE: a 130nm openroad-based tapeout-proven flow. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 110, 6 pages. doi:10.1145/3400302.3415735
[140] Kaveh Shamsi and Guangwei Zhao. 2022. An Oracle-Less Machine-Learning Attack against Lookup-Table-based Logic Locking. In *Proceedings of the Great Lakes Symposium on VLSI 2022* (Irvine, CA, USA) *(GLSVLSI '22)*. Association for Computing Machinery, New York, NY, USA, 133–137. doi:10.1145/3526241.3530377
[141] Zhengyuan Shi, Min Li, et al. 2022. DeepTPI: Test Point Insertion with Deep Reinforcement Learning. In *2022 IEEE International Test Conference (ITC)*. 194–203. doi:10.1109/ITC50671.2022.00027
[142] Zhengyuan Shi, Min Li, et al. 2023. SATformer: Transformer-Based UNSAT Core Learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–4. doi:10.1109/ICCAD57390.2023.10323731
[143] Zhengyuan Shi, Zeju Li, et al. 2025. ForgeEDA: A Comprehensive Multimodal Dataset for Advancing EDA. arXiv:2505.02016 [cs.AR] https://arxiv.org/abs/2505.02016
[144] Zhengyuan Shi, Chengyu Ma, et al. 2025. DeepCell: Multiview Representation Learning for Post-Mapping Netlists. arXiv:2502.06816 [cs.LG] https://arxiv.org/abs/2502.06816
[145] Zhengyuan Shi, Hongyang Pan, et al. 2023. DeepGate2: Functionality-Aware Circuit Representation Learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323798
[146] Zhengyuan Shi, Ziyang Zheng, et al. 2024. DeepGate3: Towards Scalable Circuit Representation Learning. arXiv:2407.11095 [cs.LG] https://arxiv.org/abs/2407.11095
[147] Mathias Soeken, Winston Haaswijk, et al. 2018. Practical exact synthesis. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 309–314. doi:10.23919/DATE.2018.8342027
[148] Jialin Song, Aidan Swope, et al. 2024. CircuitVAE: Efficient and Scalable Latent Circuit Optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 302, 6 pages. doi:10.1145/3649329.3656543
[149] Sören Richard Stahlschmidt, Benjamin Ulfenborg, et al. 2022. Multimodal deep learning for biomedical data fusion: a review. *Briefings in Bioinformatics* 23, 2 (01 2022), bbab569. doi:10.1093/bib/bbab569 arXiv:https://academic.oup.com/bib/article-pdf/23/2/bbab569/42805085/bbab569.pdf
[150] Yang Sun and Spencer Millican. 2019. Test Point Insertion Using Artificial Neural Networks. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 253–258. doi:10.1109/ISVLSI.2019.00054
[151] Yang Sun and Spencer K. Millican. 2022. Applying Artificial Neural Networks to Logic Built-in Self-test: Improving Test Point Insertion. *Journal of Electronic Testing* 38 (2022), 339–352. doi:10.1007/s10836-022-06016-9
[152] Richard S Sutton, Andrew G Barto, and others. 1998. *Reinforcement learning: An introduction.* Vol. 1. MIT press Cambridge.
[153] Veronika Thost and Jie Chen. 2021. Directed Acyclic Graph Neural Networks. arXiv:2101.07965 [cs.LG] https://arxiv.org/abs/2101.07965
[154] Herve Jacques Touati. 1990. *Performance-oriented technology mapping.* University of California, Berkeley.

[155] Dimitris Tsaras, Antoine Grosnit, et al. 2024. ShortCircuit: AlphaZero-Driven Generative Circuit Design. https://openreview.net/forum?id=KjTh5C0z7Y

[156] Petar Veličković, Guillem Cucurull, et al. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML] https://arxiv.org/abs/1710.10903

[157] Wenxi Wang, Yang Hu, et al. 2024. NeuroBack: Improving CDCL SAT Solving using Graph Neural Networks. arXiv:2110.14053 [cs.AI] https://arxiv.org/abs/2110.14053

[158] Ziyi Wang, Chen Bai, et al. 2025. FGNN2: A Powerful Pretraining Framework for Learning the Logic Functionality of Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 1 (2025), 227–240. doi:10.1109/TCAD.2024.3434464

[159] Ziyi Wang, Chen Bai, et al. 2022. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 61–66. https://doi.org/10.1145/3489517.3530410

[160] Zhihai Wang, Lei Chen, et al. 2023. A Circuit Domain Generalization Framework for Efficient Logic Synthesis in Chip Design. arXiv:2309.03208 [cs.AR] https://arxiv.org/abs/2309.03208

[161] Zhihai Wang, Jie Wang, et al. 2024. Can Symbolic Regression of Boolean Functions Boost Logic Synthesis? https://openreview.net/forum?id=OzwGZP8h2A

[162] Zhihai Wang, Jie Wang, et al. 2024. Towards Next-Generation Logic Synthesis: A Scalable Neural Circuit Generation Framework. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., 99202–99231. https://proceedings.neurips.cc/paper_files/paper/2024/file/b3ac808c09f98444090a8f6c2d4bd1dc-Paper-Conference.pdf

[163] Shaoqi Wei, Kohei Shiotani, et al. 2023. Test Point Selection Using Deep Graph Convolutional Networks and Advantage Actor Critic (A2C) Reinforcement Learning. In *2023 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*. 1–6. doi:10.1109/ITC-CSCC58803.2023.10212888

[164] Claire Wolf. 2013. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys/.

[165] Haoyuan Wu, Haisheng Zheng, et al. 2025. Architect of the Bits World: Masked Autoregressive Modeling for Circuit Generation Guided by Truth Table. arXiv:2502.12751 [cs.LG] https://arxiv.org/abs/2502.12751

[166] Haoyuan WU, Haisheng Zheng, et al. 2025. Circuit Representation Learning with Masked Gate Modeling and Verilog-AIG Alignment. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=US9k5TXVLZ

[167] Nan Wu, Jiwon Lee, et al. 2022. LOSTIN: Logic Optimization via Spatio-Temporal Information with Hybrid Graph Models. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 11–18. doi:10.1109/ASAP54787.2022.00013

[168] Nan Wu, Yingjie Li, et al. 2023. Gamora: Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6. doi:10.1109/DAC56929.2023.10247828

[169] Keyulu Xu, Weihua Hu, et al. 2019. How Powerful are Graph Neural Networks? arXiv:1810.00826 [cs.LG] https://arxiv.org/abs/1810.00826

[170] Lin Xu, Frank Hutter, et al. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research* 32 (2008), 565–606.

[171] Chenghao Yang, Zhongda Wang, et al. 2023. The prediction of the quality of results in Logic Synthesis using Transformer and Graph Neural Networks. arXiv:2207.11437 [cs.AR] https://arxiv.org/abs/2207.11437

[172] Chenghao Yang, Yinshui Xia, et al. 2022. Logic Synthesis Optimization Sequence Tuning Using RL-Based LSTM and Graph Isomorphism Network. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 8 (2022), 3600–3604. doi:10.1109/TCSII.2022.3168344

[173] Shuwen Yang, Zhihao Yang, et al. 2022. Versatile Multi-stage Graph Neural Network for Circuit Representation. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Inc., 20313–20324. https://proceedings.neurips.cc/paper_files/paper/2022/file/7fa548155f40c014372146be387c4f6a-Paper-Conference.pdf

[174] Zhaohui Yang, Yinshui Xia, et al. 2023. Attention-Based Mechanism For Technology Mapping Optimization. In *2023 China Semiconductor Technology International Conference (CSTIC)*. 1–3. doi:10.1109/CSTIC58779.2023.10219217

[175] Rozhin Yasaei, Shih-Yuan Yu, et al. 2021. GNN4IP: Graph Neural Network for Hardware Intellectual Property Piracy Detection. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 217–222. doi:10.1109/DAC18074.2021.9586150

[176] Zhitao Ying, Dylan Bourgeois, et al. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32 (2019).

[177] Zhitao Ying, Jiaxuan You, et al. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/e77dbaf6759253c7c6d0efc5690369c7-Paper.pdf

[178] Emre Yolcu and Barnabas Poczos. 2019. Learning Local Search Heuristics for Boolean Satisfiability. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/

paper/2019/file/12e59a33dea1bf0630f46edfe13d6ea2-Paper.pdf
[179] Cunxi Yu. 2025. Mapping Fusion: Improving FPGA Technology Mapping with ASIC Mapper. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*. 1–7. doi:10.1109/MLCAD65511.2025.11189128
[180] Cunxi Yu, Houping Xiao, et al. 2018. Developing synthesis flows without human knowledge. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.
[181] Cunxi Yu and Wang Zhou. 2020. Decision Making in Synthesis cross Technologies using LSTMs and Transfer Learning. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD* (Virtual Event, Iceland) *(MLCAD '20)*. Association for Computing Machinery, New York, NY, USA, 55–60. doi:10.1145/3380446.3430638
[182] Jianyong Yuan, Peiyu Wang, et al. 2023. EasySO: Exploration-enhanced Reinforcement Learning for Logic Synthesis Sequence Optimization and a Comprehensive RL Environment. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323973
[183] Hanqing Zeng, Hongkuan Zhou, et al. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. *CoRR* abs/1907.04931 (2019). arXiv:1907.04931 http://arxiv.org/abs/1907.04931
[184] Wei Zeng, Azadeh Davoodi, et al. 2021. Sampling-Based Approximate Logic Synthesis: An Explainable Machine Learning Approach. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD51958.2021.9643484
[185] Muhan Zhang, Shali Jiang, et al. 2019. D-VAE: A Variational Autoencoder for Directed Acyclic Graphs. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/e205ee2a5de471a70c1fd1b46033a75f-Paper.pdf
[186] Wenjie Zhang, Zeyu Sun, et al. 2020. NLocalSAT: Boosting Local Search with Solution Prediction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-PRICAI-2020)*. International Joint Conferences on Artificial Intelligence Organization, 1177–1183. doi:10.24963/ijcai.2020/164
[187] Yanqing Zhang, Haoxing Ren, et al. 2020. GRANNITE: Graph Neural Network Inference for Transferable Power Estimation. In *Proceedings of the 57th ACM/IEEE Design Automation Conference*. IEEE, 1–6. doi:10.1109/DAC18072.2020.9218643
[188] Ziwei Zhang and Yang Zhang. 2021. Elimination mechanism of glue variables for solving sat problems in linguistics. In *The Asian Conference on Language*. 147–167.
[189] Guangwei Zhao and Kaveh Shamsi. 2022. Graph Neural Network based Netlist Operator Detection under Circuit Rewriting. In *Proceedings of the Great Lakes Symposium on VLSI 2022* (Irvine, CA, USA) *(GLSVLSI '22)*. Association for Computing Machinery, New York, NY, USA, 53–58. doi:10.1145/3526241.3530330
[190] Hongfan Zhao, Fan Yang, et al. 2024. A Graph AutoEncoder Approach for Fault Prediction in Test Pattern Generation. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. 462–467. doi:10.1109/ISEDA62518.2024.10618040
[191] Haisheng Zheng, Zhuolun He, et al. 2024. LSTP: A Logic Synthesis Timing Predictor. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 728–733. doi:10.1109/ASP-DAC58780.2024.10473925
[192] Ziyang Zheng, Shan Huang, et al. 2025. DeepGate4: Efficient and Effective Representation Learning for Circuit Design at Scale. arXiv:2502.01681 [cs.LG] https://arxiv.org/abs/2502.01681
[193] Guanglei Zhou and Jason H. Anderson. 2023. Area-Driven FPGA Logic Synthesis Using Reinforcement Learning. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '23)*. Association for Computing Machinery, New York, NY, USA, 159–165. doi:10.1145/3566097.3567894
[194] Xinyi Zhou, Xing Li, et al. 2024. SeaDAG: Semi-autoregressive Diffusion for Conditional Directed Acyclic Graph Generation. arXiv:2410.16119 [cs.LG] https://arxiv.org/abs/2410.16119
[195] Xinyi Zhou, Xing Li, et al. 2025. Circuit Synthesis based on Hierarchical Conditional Diffusion. In *Proceedings of the Great Lakes Symposium on VLSI 2025 (GLSVLSI '25)*. Association for Computing Machinery, New York, NY, USA, 634–640. doi:10.1145/3716368.3735204
[196] Keren Zhu, Mingjie Liu, et al. 2020. Exploring logic optimizations with reinforcement learning and graph convolutional network. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*. 145–150.
[197] Linyu Zhu and Xinfei Guo. 2023. Delay-Driven Physically-Aware Logic Synthesis with Informed Search. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. 327–335. doi:10.1109/ICCD58817.2023.00057
[198] Zhufei Chu et al. 2019. ALSO: Advanced Logic Synthesis and Optimization Tool. https://gitee.com/zfchu/also.
[199] Dongsheng Zuo, Yikang Ouyang, et al. 2023. RL-MUL: Multiplier Design Optimization with Deep Reinforcement Learning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6. doi:10.1109/DAC56929.2023.10247941

Received 20 February 2027; revised 12 March 2027; accepted 5 June 2027