Machine Learning in EDA: When and How

Bei Yu
Chinese University of Hong Kong

***Abstract*—Machine learning is a powerful technique that can derive knowledge from large data set, and provide prediction and modeling. Since VLSI chip designs have extremely high complexity and gigantic data, recently there has been a surge in applying and adapting machine learning to accelerate the design closure. In this paper, we will discuss when and how to apply machine learning in Electronic Design Automation (EDA) improving the efficiency and quality of the design process. Furthermore, we highlight distinct challenges in EDA, including improved netlist representation, advanced timing modeling, netlist-layout multimodality, and constrained AIGC.**

I. Machine Learning in EDA

Over the past few decades, there has been a noticeable trend towards increasing standardization and complexity in the field of Electronic Design Automation (EDA). The contemporary chip design flow can be divided into two distinct parts: firstly, the front-end EDA, encompassing architectural design, high-level synthesis, and logic synthesis; and secondly, the back-end EDA, comprising placement and routing. As semiconductor technology continues to advance, the scale of integrated circuits has grown exponentially, presenting significant challenges to the scalability and reliability of the chip design flow.

Several recent survey papers have been published on the topic of ML-EDA. Rapp *et al.* [1] presented a comprehensive survey of the application of machine learning in the optimization and exploration strategies of integrated circuits, along with trends in the employed ML algorithms. Huang *et al.* [2] categorized existing ML studies in the EDA field into four distinct categories: decision-making, performance prediction, black-box optimization, and automated design, ordered by increasing degree of automation. Chen *et al.* [3] summarized the state-of-the-art research in ML-EDA, utilizing a taxonomy of ML methodologies, and offered potential insights from previously resolved EDA problems. It is worth noting that their focus is primarily on applied ML algorithms in recent years, which complements our survey that specifically examines the unique challenges encountered in EDA.

In this section, we aim to provide a comprehensive survey on the application of ML methods in each stage of the EDA flow, as illustrated in Figure 1. We will commence by discussing the microarchitecture design of a processor, which aims to define the implementation of an instruction set architecture (ISA). Due to the vast design space, an efficient design space exploration (DSE) technique, coupled with feature engineering, is promising for microarchitecture design. One notable approach in this domain is BOOM-Explorer [4], which utilizes a novel Gaussian process model with deep kernel learning functions

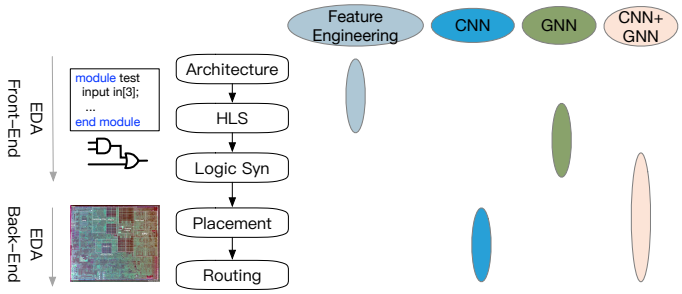

Fig. 1 Typical ML methodologies in different EDA stages.

to characterize the relevant features. This enables BOOM-Explorer to explore microarchitecture designs that strike an optimal balance between power consumption and performance, while significantly reducing the time required for DSE.

High-level synthesis (HLS) transforms a high-level specification of an integrated circuit (IC) into a register-transfer level (RTL) description. However, the synthesis process can be time-consuming, especially for large-scale systems. To address this challenge, machine learning algorithms, including feature engineering and graph neural networks (GNN), have been employed to accelerate HLS. For instance, Sun *et al.* [5] developed a novel correlated multiobjective and multi-fidelity Gaussian process (CGP) model to effectively handle the relationships among various design objectives, thereby improving the efficiency of HLS. Additionally, Ferretti *et al.* [6] proposed the use of graph representation learning from software specifications. Fine-tuned with a few-shot learning approach, the learned model enables effective DSE after a short training period.

Following high-level synthesis, logic synthesis is responsible for transforming the RTL description of a circuit into a gate-level representation in the target technology. Notably, both GNN and convolutional neural networks (CNN) have been leveraged to expedite the logic synthesis process. Xu *et al.* [7] introduced SNS, which combines GNN and CNN to predict the area, power, and timing of a wide range of designs. By leveraging this approach, the delay in obtaining synthesis results can be significantly reduced. Wang *et al.* [8] developed a novel framework for netlist representation learning based on contrastive learning (CL). This framework extracts the fundamental logic functionality of netlists using a customized GNN architecture designed specifically for circuit representation learning, thereby improving efficiency in logic synthesis.

Machine-learning techniques, such as CNN and combinations of CNN and GNN, have also found applications in the back-end EDA flow, i.e., placement and routing. To accelerate placement, DREAMPlace [9] tackles the analytical placement problem by analogizing it to training a neural network, which enables more efficient placement algorithms. Liu *et al.* [10]

979-8-3503-0955-3/23/$31.00 ©2023 IEEE

employed a CNN to predict congestion hotspots, which are then integrated into a placement engine to achieve more route-friendly results. Routing is typically the most time-consuming task in physical design, and it often requires a combination of ML models and traditional algorithms for effective solutions. For instance, Qu *et al.* [11] proposed a reinforcement learning (RL)–based algorithm that learns an ordering policy to minimize design rule check violations based on net features. To reduce turnaround time at the pre-routing stage, Liu *et al.* [12] designed a concurrent learning-assist early-stage timing optimization framework called TSteiner. They utilize a customized GNN to obtain sign-off timing optimization gradients, which guide the refinement of Steiner points. These approaches demonstrate the potential of combining ML methodologies with traditional algorithms in back-end EDA tasks.

II. How Machine Learning Integrated

A. Methodologies Perspective

There are significant disparities between traditional EDA methodologies and ML approaches. Traditional EDA encompasses techniques such as placement, routing, synthesis, simulation, and more. These methods demonstrate robustness through the analysis of optimality in known problems. They require less training data, exhibit solvability in known problem domains, and offer good interpretability. However, when confronted with dynamic and complex problems, traditional EDA methods may oversimplify the problem, leading to suboptimal solutions.

In contrast, machine learning methods encompass supervised learning [13], unsupervised learning [8], [14], and reinforcement learning techniques [15]. Machine learning methods are amenable to parallel computation using GPUs [16], [17], facilitating the efficient design and end-to-end training for complex problems. Leveraging data, machine learning methods demonstrate the ability to solve a wide range of problems. However, they are susceptible to overreliance on data, potentially underutilizing the inherent mechanisms and characteristics of the problem.

B. Task Type Perspective

Over time, there have been noticeable trends in the application of machine learning in the field of EDA.

Fig. 2 categorizes articles from recent MLCAD conferences, providing insights into the changing proportions of research on different data types. Notably, studies on tabular data consistently maintained a high proportion, accounting for 50.00% in 2019 and 68.75% in 2021. Furthermore, there is an increasing trend in research on image data, with proportions reaching 24.00% in 2020 and 26.09% in 2022. We also observed significant growth in research related to graph data, with respective increases of 12.00% in 2019 and 13.50% in 2022 compared to the previous years. In 2022, graph data research accounted for 26.09%, positioning it as an equally important focus as image data. While the proportion of research on textual data is relatively low, its significance should not be underestimated.

We next delve into the different purposes of machine learning algorithms applied to EDA, as observed in recent MLCAD conferences. The purposes are categorized as shown in Figure

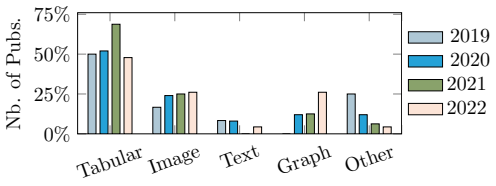

Fig. 2 Recent years show a noticeable trend toward a high proportion of tabular data, accompanied by an increase in image and graph data.

3. Decision-making emerges as the primary focus in recent years, with proportions of 50.00% in 2019, 56.00% in 2020, and 34.78% in 2022. Classification and regression are identified as other essential research objectives, with relatively balanced proportions of 37.50% each in 2019 and 2021. However, there has been a notable increase in the proportion of regression, reaching 32.00% in 2020 and 52.17% in 2022. This suggests a growing focus on utilizing machine learning for classification, regression analysis, prediction, and modeling in EDA applications. Lastly, generation represents a relatively small research area, with proportions peaking at 8.70% in 2022.

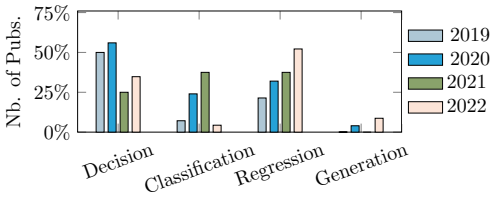

Fig. 3 The observed trend reflects a strong interest for using machine learning for decision support, classification, and regression analysis.

C. Learning Closure Perspective

In traditional machine learning approaches, data is typically divided into training and testing sets for model evaluation. However, this approach has limitations when applied to EDA scenarios. To successfully introduce AI-accelerated EDA technologies to the market, a different machine learning paradigm, as shown in Fig. 4, is required.

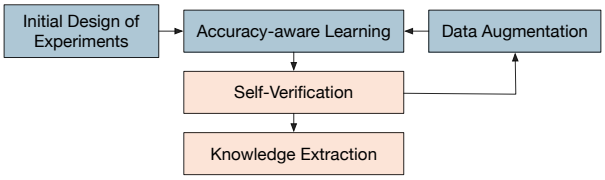

Fig. 4 Adaptive integrated machine learning paradigm.

At the beginning of this process, it is crucial to design initial experiments and establish a data acquisition loop for the targeted EDA application. The subsequent training process aims to achieve the desired accuracy, which aims to dynamically adjust the training based on the accuracy requirements of the model to align the preset requirements. Additionally, this process is accompanied by a self-validation phase that takes into account out-of-distribution factors, such as PVT variations, tails of Monte Carlo distributions, and more. Based on the models' accuracy and robustness, further data augmentation is

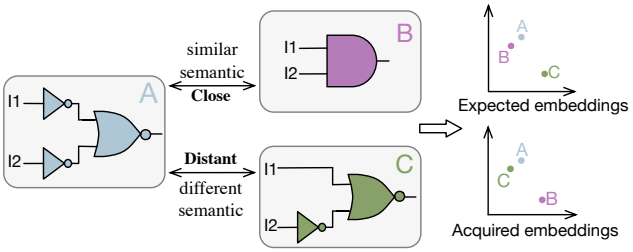

Fig. 5 Illustration of the main challenge for netlist representation learning.

performed to enhance the model's precision and generalization. After completing the closed-loop training process, the extraction of relevant knowledge becomes paramount. The extracted knowledge is then fed back to design teams to further improve the design process.

By adopting this machine learning paradigm, a wide range of chip design challenges can be effectively addressed, surpassing the traditional "train and test" paradigm.

III. Unique Challenges in EDA

Although so many machine learning algorithms have been deployed in various EDA stages, there are still some unique challenges that need to be solved.

A. Better Netlist Representations

Due to the graph nature of netlists, the emerging Graph Neural Networks (GNNs) have become the top choice for netlist representation learning [18]. A conventional GNN [19] follows an iterative neighborhood aggregation scheme to capture the structural information within nodes' neighborhoods. In recent years, we have seen a wide application of GNN in many netlist-level tasks, e.g., testability analysis [20], reverse engineering [14], [21], power estimation [22], etc. While these GNN-driven works have achieved promising results compared with traditional methods, they are far from competent for learning high-quality netlist representations, as pointed out by previous studies [8]. The main challenge faced by the conventional GNN methods is their limited generalization capacity, stemming from the inherent instability of netlist structures. In particular, the netlist (graph) structure of a given circuit might vary greatly with respect to different technology nodes, cell libraries, or even logic synthesis tools.

Fig. 5 gives an example to illustrate the limitation of conventional GNNs when dealing with netlists, given the fact that semantic (logic functionality) and structural information of netlists may conflict with each other. We consider three distinct netlists, denoted as A, B, and C, respectively. In this context, both A and B implement the same function and share akin semantics. As a consequence, they should ideally manifest proximity within the representation space. However, structural methods would push their representations apart, driven by their disparate structures. Conversely, A and C implement different functions, thus diverging in terms of their underlying semantics. Nevertheless, their representations would be pulled close by structural methods based on their similar structures.

As evidenced by the above example, the main challenge at hand centers around developing methodologies that can transcend the constraint of structural instability, enabling the acquisition of netlist representations that possess broader applicability and robustness. Encouragingly, recent endeavors have been directed towards enhancing the quality of netlist representations with an emphasis on better generalization capabilities. For instance, Wang *et al.* [8] introduce a novel self-supervised netlist representation learning flow that aspires to learn universal netlist knowledge. By utilizing contrastive learning, netlists/gates with similar functionality (semantic) will be drawn closer in the representation space, while those with distinct functionality will be pushed away. The results presented by them demonstrate the notable superiority of functionality-based netlist representations compared with structural ones, particularly when generalizing to unseen data. Similarly, DeepGate [24] utilizes signal probability (i.e., the probability of being logic '1') for every gate as supervision to learn the logic functionality of netlists. To bolster the generalization potential of their model, the authors further transform the netlists into a unified form, and-invertor graphs (AIGs). Nevertheless, representations learned through signal probability can not model the relative distance between different logic functions. TABLE I summarizes the comparison between different netlist representation learning methods. In sum, recent advances showcase a promising trajectory toward more adaptive and robust netlist representations.

TABLE I Comparison among netlist representation methods

Characteristic	Deep	DAG	AIG	Functional	Relative-similarity
ShapeHashing [23]	✗	✓	✗	✗	✗
GraphSage [19]	✓	✗	✗	✗	✗
ABGNN [14]	✓	✓	✗	✗	✗
FGNN [8]	✓	✓	✗	✓	✓
DeepGate [24]	✓	✓	✓	✓	✗

B. Timing Modeling

Fig. 6 illustrates one timing path where one signal propagates from the startpoint to the endpoint. A timing path contains many cells and wires. During timing analysis, it is necessary to give accurate cell and wire delay results in a fast way. Traditionally, cell timing is calculated based on look-up-tables, including nonlinear cell delay model (NLDM) and current source model (CSM) [25], and wire timing is computed based on analytical models, including Elmore model [26] and D2M model [27]. However, the accuracy and efficiency of traditional methods cannot meet timing sign-off requirements in advanced technologies. Some simple machine learning models, such as XGBoost and random forest, helped to solve some timing modeling problems, such as cell delay modeling [28], wire delay modeling [29], path-based timing analysis [30], and routing-free timing analysis [31]. However, it is still hard for simple learning models to capture structural information, which limits their accuracy [32].

To solve the issue, timing modeling has stepped into the graph-learning era [32]–[35]. Netlists are described as graphs where cells are nodes and wires are edges. Popular graph learning methods are used to learn information from circuit structure through aggregating information from local neighbors. They can achieve node embedding and graph embedding. For

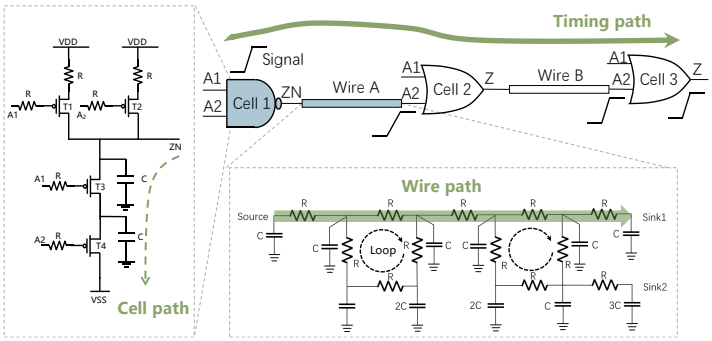

Fig. 6 One example of the timing path in the netlist, cell (cell path) and wire (wire path).

timing modeling, the rich information of timing paths is much more important but is difficult to learn via current graph learning methods [34], [35].

Timing paths are composed of cells and wires. For cell timing modeling and wire timing modeling, there are few paths in them. As shown in Fig. 6, the timing path in cells, called cell path, is the timing arc containing the charging path when PMOS is on and the discharging paths when NMOS is on; In addition, the timing path in wires, called wire path, is the path for a signal to propagate from the wire source to the target wire sink. The path number in one cell and one wire is much smaller than that in large-scale netlists. The limited number of cell paths and wire paths opens a door for graph learning to collect their path information with high efficiency. HGAT [36] can aggregate heterogeneous information from transistors, capacitors and resistors on different cell paths simultaneously. GNNTrans [37] also embed wire path through combining local wire structure information learned from GNNs and global relationships among all elements minded by the transformer.

For large-scale netlists, the number of timing paths in them exponentially increases with numbers of cells and wires. Conventional graph learning methods have extremely low efficiency and memory issues in embedding all paths in netlists. Besides, many layers need to be stacked to learn information for long netlist paths. The deep model causes an over-smoothing issue, which significantly degrades timing modeling accuracy [34], [37]. Thus, achieving path embedding on the netlist level is still an open challenge for timing modeling in ML for EDA topics.

C. Netlist+Layout: Multimodality

Diverse deep-learning-based approaches have been proposed to replace the time-consuming engines in EDA tools to predict routing congestion [10], [38]–[41], design rule violation [42]–[44], timing [13], [45], [46], etc. To improve accuracy, most existing methods use either vision models with geometric layout features or graph models with logical netlist connections. However, these methods have several common issues. First, they do not fully explore how to fuse layout and netlist features from different modalities. Thus, they cannot effectively combine the information from cell locations and net connectivity. Second, they only use local information and ignore long-range interactions. For example, vision-based models use convolutional layers to extract local features for congestion prediction, but they

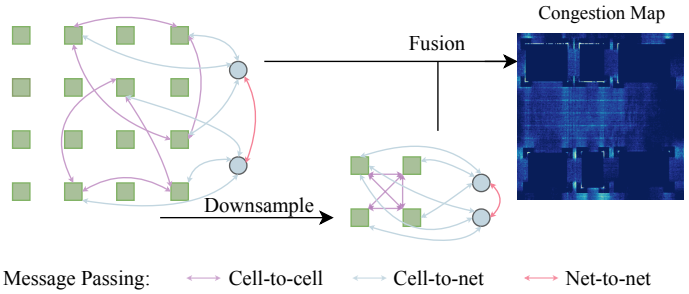

Fig. 7 Illustration of a possible multi-modality fusion solution. It enables global information aggregation by utilizing hierarchical feature maps and explicitly models the routing demand via net-to-net message passing.

TABLE II Comparison among routability prediction methods

Characteristic	RUDY*	Macro	Routing-Free	Cell-to-cell	Cell-to-net	Multi-scale
RouteNet [42]	✓	✓	✗	✗	✗	✗
GAN [38]	✓	✓	✓	✗	✗	✗
NAS [49]	✓	✓	✓	✗	✗	✗
Cross-Graph [50]	✗	✗	✓	✓	✗	✗
LHNN [40]	✓	✗	✓	✓	✓	✗
PGNN [43]	✓	✗	✓	✓	✗	✗
CircuitGNN [51]	✓	✗	✓	✓	✓	✗
Lay-Net [47]	✓	✓	✓	✓	✓	✓

*: Any network that is aware of routability features are considered as RUDY-Aware.

lack a global view of the layout. For graph-based methods, the over-smoothing problem of GNN prevents them from collecting long-range information. Third, existing GNN models overlook the interactions arising from the overlaps of nets, which is a crucial factor contributing to non-ideal results like routing congestion. Even though the long-range connections can be established according to the netlist, the cell-to-cell or cell-to-net links in existing approaches [40] cannot directly model the physical interactions in GNNs. These limitations highlight the demand for multimodal models customized to EDA problems.

Recently, we noticed some efforts trying to address the above challenges. A representative work, Lay-Net [47], is illustrated in Fig. 7. It embeds message passing in vision-based models to enable long-range information collection via hierarchical feature maps and enrich graph message passing via improved connection types. Specifically, Lay-Net conducts the netlist-based message passing on the output of a vision-based block, serving as a multi-modality fusion mechanism. For long-range information, it utilizes Swin Transformer [48] to construct hierarchical feature maps based on layout features, which models an image in various scales with patch merging and extracts multi-scale features with self-attention in shifted windows. Furthermore, the GNN model is enhanced by introducing a heterogeneous graph neural network structure that enables cell-to-cell, cell-to-net, and net-to-net message passing.

TABLE II presents the difference between different models for routability prediction, including congestion prediction, design rule violation (DRV) prediction, etc. Most methods are RUDY-aware since routability-based features are essential for congestion prediction. Macro-aware feature is important because we observe that congestion frequently appears around macros. Routing-free is also a common feature, which means

that a method does not rely on the time-consuming trial global routing process. Among these methods, Lay-Net [47] shows superior performance, highlighting the importance of layout-netlist information fusion and multi-scale feature extraction in congestion prediction.

D. Constrained AIGC

Differing from the general concept of artificial intelligence generated content (AIGC), the generated content within the EDA domain is typically subject to additional design rules. Evaluating the quality of models relies significantly on the legality of the output. A prime example of constrained AIGC in EDA pertains to the generation of layout patterns. The establishment of reliable layout pattern libraries serves as the cornerstone for diverse designs for manufacturability research. With the escalating demand for layout patterns in lithography design applications based on machine learning [52]–[54], constructing a feasible large-scale pattern library could prove highly time-consuming due to the extended logic-to-chip design cycle. Recent literature has proposed several learning-based methods for generating layout patterns, such as [55]–[58]. To fit the latent distribution of layout patterns and generate novel instances, famous generative models in computer vision domain have been introduced in pattern generation task. However, two predominant constraints differentiate layout pattern generation from conventional image generation.

The first significant constraint pertains to the discrete nature of layout patterns, as depicted in Fig. 8. In a layout pattern, the state of each pixel is binary, while prevailing image generation techniques are designed for continuous state spaces. To transform a continuous model output to a layout pattern, some existing works [55], [56] transform these into layout topologies through binary truncation upon generating continuous examples. However, such truncation could potentially compromise the model's capacity. Since the details of model prediction is removed in the truncation process. To address this concerns, DiffPattern [58] introduces a practical framework for layout pattern generation. Through the application of a discrete diffusion model, DiffPattern confines each entry's state within a pre-defined discrete state space. As a result, DiffPattern can directly generate discrete layout patterns without clipping.

On a different note, the generated layout patterns are required to follow the design rules, which makes the layout pattern generation more challenging. To prevent the production of illicit layout patterns that contravene design rules, [56], [57] derive latent regularization from the training dataset. However, the implicit constraint learned from this training set might lack flexibility and reliability. Beyond the inconvenience of needing to train a new model on a specific dataset that adheres to updated design rules, a significant proportion of the generated patterns violate these rules. Addressing these concerns, DiffPattern [58] devises a nonlinear system capable of identifying a legal solution for each topology matrix. This system can be easily adjusted to accommodate various design rules.

While the realm of constrained AIGC in the context of EDA has experienced substantial exploration in recent literature, the pursuit of a more robust constrained AIGC approach within EDA remains a persistently challenging and evolving endeavor.

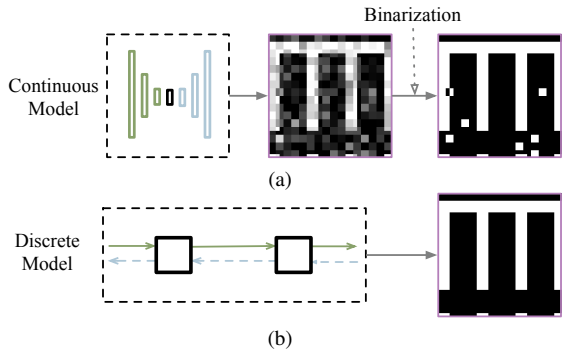

Fig. 8 Illustration of the discrete state space constraint in layout pattern generation. (a) The binarization on continuous model output may lead to information loss; (b) Discrete model can output discrete samples directly.

IV. Conclusion and Future Direction

Recent advancements have witnessed the integration of ML into EDA, a merger that has promised and delivered notable improvements in the design flow. This incorporation has been marked by successful outcomes in classification, detection, and design space exploration challenges. Despite the advancements and numerous ML algorithms introduced into EDA, the path forward still presents unique obstacles, like the development of better netlist representations and addressing issues related to timing modeling, netlist-layout multimodality, and constrained content generation.

One future direction is applying large language models (LLMs) into the EDA flow, harnessing their analytical capabilities to optimize chip design processes, and revolutionize the way EDA flows are conceptualized and implemented. Interested readers may refer to [59], [60] for more discussions and explorations.

Acknowlegement

The author thanks many students and collaborators, who have helped to develop the works and perspectives given in this paper: Guojin Chen, Hongduo Liu, Zixiao Wang, Ziyi Wang, Peng Xu, Yuyang Ye, Yu Zhang, Su Zheng.

References

[1] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Z. Pan, M. Wolf, and J. Henkel, "MLCAD: A survey of research in machine learning for CAD keynote paper," *IEEE TCAD*, vol. 41, no. 10, pp. 3162–3181, 2021.

[2] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong *et al.*, "Machine learning for electronic design automation: A survey," *ACM TODAES*, vol. 26, no. 5, pp. 1–46, 2021.

[3] T. Chen, G. L. Zhang, B. Yu, B. Li, and U. Schlichtmann, "Machine learning in advanced IC design: A methodological survey," *IEEE MDAT*, vol. 40, no. 1, pp. 17–33, 2022.

[4] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework," in *Proc. ICCAD*, 2021.

[5] Q. Sun, T. Chen, S. Liu, J. Miao, J. Chen, H. Yu, and B. Yu, "Correlated multi-objective multi-fidelity optimization for hls directives design," in *Proc. DATE*, 2021.

[6] L. Ferretti, A. Cini, G. Zacharopoulos, C. Alippi, and L. Pozzi, "Graph neural networks for high-level synthesis design space exploration," *ACM TODAES*, vol. 28, no. 2, pp. 1–20, 2022.

[7] C. Xu, C. Kjellqvist, and L. W. Wills, "Sns's not a synthesizer: A deep-learning-based synthesis predictor," in *Proc. ISCA*, 2022.

[8] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proc. DAC*, 2022.

[9] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. DAC*, 2019.

[10] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proc. DATE*, 2021.

[11] T. Qu, Y. Lin, Z. Lu, Y. Su, and Y. Wei, "Asynchronous reinforcement learning framework for net order exploration in detailed routing," in *Proc. DATE*, 2021.

[12] S. Liu, Z. Wang, F. Liu, Y. Lin, B. Yu, and M. Wong, "Concurrent sign-off timing optimization via deep steiner points refinement," in *Proc. DAC*, 2023.

[13] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Realistic sign-off timing prediction via multimodal fusion," in *Proc. DAC*, 2023.

[14] Z. He, Z. Wang, C. Bai, H. Yang, and B. YU, "Graph learning-based arithmetic block identification," in *Proc. ICCAD*, 2021.

[15] Z. Pei, F. Liu, Z. He, G. Chen, H. Zheng, K. Zhu, and B. Yu, "AlphaSyn: Logic synthesis optimization with efficient monte carlo tree search," in *Proc. ICCAD*, 2023.

[16] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. DATE*, 2021.

[17] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. ICCAD*, 2021.

[18] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong *et al.*, "Machine learning for electronic design automation: A survey," *ACM TODAES*, vol. 26, no. 5, pp. 1–46, 2021.

[19] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017.

[20] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," in *Proc. DAC*, 2019.

[21] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE TCAD*, vol. 41, no. 8, pp. 2435–2448, 2021.

[22] Y. Zhang, H. Ren, and B. Khailany, "Grannite: Graph neural network inference for transferable power estimation," in *Proc. DAC*, 2020.

[23] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia, "Wordrev: Finding word-level structures in a sea of bit-level gates," in *Proc. HOST*, 2013.

[24] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proc. DAC*, 2022.

[25] Synopsys, "PrimeTime user guide," https://www.synopsys.com/cgi-bin/imp/pdfdla/pdfr1.cgi?file=primetime-wp.pdf, 2023.

[26] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[27] C. J. Alpert, A. Devgan, and C. Kashyap, "A two moment RC delay metric for performance optimization," in *Proc. ISPD*, 2000.

[28] S. M. Ebrahimipour, B. Ghavami, H. Mousavi, M. Raji, Z. Fang, and L. Shannon, "Aadam: a fast, accurate, and versatile aging-aware cell library delay model using feed-forward neural network," in *Proc. ICCAD*, 2020.

[29] H.-H. Cheng, I. H.-R. Jiang, and O. Ou, "Fast and accurate wire timing estimation on tree and non-tree net structures," in *Proc. DAC*, 2020.

[30] A. B. Kahng, U. Mallappa, and L. Saul, "Using machine learning to predict path-based slack from graph-based timing analysis," in *Proc. ICCD*, 2018.

[31] D. Hyun, Y. Fan, and Y. Shin, "Accurate wirelength prediction for placement-aware synthesis through machine learning," in *Proc. DATE*, 2019.

[32] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, and G.-J. Nam, "Routing-free crosstalk prediction," in *Proc. ICCAD*, 2020.

[33] Z. Xie, R. Liang, X. Xu, J. Hu, Y. Duan, and Y. Chen, "Net2: A graph attention network method customized for pre-placement net length estimation," in *Proc. ASPDAC*, 2021.

[34] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proc. DAC*, 2022.

[35] K. K.-C. Chang, C.-Y. Chiang, P.-Y. Lee, and I. H.-R. Jiang, "Timing macro modeling with graph neural networks," in *Proc. DAC*, 2022.

[36] Y. Ye, T. Chen, Z. Wang, H. Yan, B. Yu, and L. Shi, "Fast and accurate aging-aware cell timing model via graph learning," *IEEE TCAS II*, 2023.

[37] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Fast and accurate wire timing estimation based on graph learning," in *Proc. DATE*, 2023.

[38] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proc. DAC*, 2019.

[39] W. Li, G. Chen, H. Yang, R. Chen, and B. Yu, "Learning point clouds in eda," in *Proc. ISPD*, 2021.

[40] B. Wang, G. Shen, D. Li, J. Hao, W. Liu, Y. Huang, H. Wu, Y. Lin, G. Chen, and P. A. Heng, "LHNN: Lattice hypergraph neural network for VLSI congestion prediction," in *Proc. DAC*, 2022.

[41] S. Zheng, L. Zou, S. Liu, Y. Lin, B. Yu, and M. D. F. Wong, "Mitigating distribution shift for congestion optimization in global placement," in *Proc. DAC*, 2023.

[42] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. ICCAD*, 2018.

[43] K. Baek, H. Park, S. Kim, K. Choi, and T. Kim, "Pin accessibility and routing congestion aware DRC hotspot prediction using graph neural network and U-Net," in *Proc. ICCAD*, 2022.

[44] R. Liang, H. Xiang, J. Jung, J. Hu, and G.-J. Nam, "A stochastic approach to handle non-determinism in deep learning-based design rule violation predictions," in *Proc. ICCAD*, 2022.

[45] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. DAC*, 2019.

[46] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead RC network," in *Proc. DAC*, 2022.

[47] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. D. F. Wong, "Lay-net: Grafting netlist knowledge on layout-based congestion prediction," in *Proc. ICCAD*, 2023.

[48] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. CVPR*, 2021.

[49] C.-C. Chang, J. Pan, T. Zhang, Z. Xie, J. Hu, W. Qi, C.-W. Lin, R. Liang, J. Mitra, E. Fallon, and Y. Chen, "Automatic routability predictor development using neural architecture search," in *Proc. ICCAD*, 2021.

[50] A. Ghose, V. Zhang, Y. Zhang, D. Li, W. Liu, and M. Coates, "Generalizable cross-graph embedding for gnn-based congestion prediction," in *Proc. ICCAD*, 2021.

[51] Z. Yang, D. Li, Y. Zhang, Z. Zhang, G. Song, J. Hao *et al.*, "Versatile multi-stage graph neural network for circuit representation," *Proc. NeurIPS*, vol. 35, pp. 20 313–20 324, 2022.

[52] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full-chip scale," *IEEE TCAD*, vol. 41, no. 9, pp. 3118–3131, 2022.

[53] G. Chen, Z. Pei, H. Yang, Y. Ma, B. Yu, and M. Wong, "Physics-informed optical kernel regression using complex-valued neural fields," in *Proc. DAC*, 2023.

[54] W. Zhao, X. Yao, Z. Yu, G. Chen, Y. Ma, B. Yu, and M. D. F. Wong, "AdaOPC: A self-adaptive mask optimization framework for real design patterns," in *Proc. ICCAD*, 2022.

[55] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Deepattern: Layout pattern generation with transforming convolutional auto-encoder," in *DAC*, 2019.

[56] X. Zhang, J. Shiely, and E. F. Young, "Layout pattern generation and legalization with generative learning models," in *ICCAD*, 2020.

[57] L. Wen, Y. Zhu, L. Ye, G. Chen, B. Yu, J. Liu, and C. Xu, "Layoutransformer: Generating layout patterns with transformer via sequential pattern modeling," in *ICCAD*, 2022.

[58] Z. Wang, Y. Shen, W. Zhao, Y. Bai, G. Chen, F. Farnia, and B. Yu, "Diffpattern: Layout pattern generation via discrete diffusion," *arXiv preprint arXiv:2303.13060*, 2023.

[59] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design," in *Proc. MLCAD*, 2023.

[60] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "ChatEDA: A Large Language Model Powered Autonomous Agent for EDA," in *Proc. MLCAD*, 2023.