

Transferable Presynthesis PPA Estimation for RTL Designs With Data Augmentation Techniques

Wenji Fang^{1b}, Graduate Student Member, IEEE, Yao Lu^{1b}, Shang Liu^{1b}, Qijun Zhang^{1b}, Ceyu Xu, Lisa Wu Wills, Hongce Zhang^{1b}, Member, IEEE, and Zhiyao Xie^{1b}, Member, IEEE

Abstract—In modern VLSI design flow, evaluating the quality of register-transfer level (RTL) designs involves time-consuming logic synthesis using electronic design automation tools, a process that often slows down early optimization. While recent machine learning (ML) solutions offer some advancements, they typically struggle with maintaining high accuracy across any given RTL design. In this work, we propose an innovative transferable presynthesis power, performance, and area (PPA) estimation framework named MasterRTL. It first converts the hardware description language code to a new bit-level design representation named the simple operator graph (SOG). By only adopting single-bit simple operators, this SOG proves to be a general representation that unifies different design types and styles. The SOG is also more similar to the target gate-level netlist, reducing the gap between the RTL representation and netlist. In addition to the new SOG representation, MasterRTL proposes new ML methods for the RTL-stage modeling of timing, power, and area separately. Compared with the state-of-the-art solutions, the experiment on a comprehensive dataset with 90 different designs shows accuracy improvement by 0.33, 0.22, and 0.15 in correlation for total negative slack (TNS), worst negative slack (WNS), and power, respectively. Besides the prediction of the synthesis results, MasterRTL also excels in accurately predicting layout-stage PPA based on the RTL designs and in adapting across different technology nodes and process corners. Furthermore, we investigate two effective data augmentation techniques: 1) a graph generation method and 2) a large language model (LLM)-based approach. Our results validate the effectiveness of the generated RTL designs in mitigating the data shortage challenges.

Index Terms—Data augmentation, power modeling, register-transfer level (RTL), timing analysis.

Manuscript received 22 January 2024; revised 22 April 2024 and 16 June 2024; accepted 20 June 2024. Date of publication 1 July 2024; date of current version 26 December 2024. This work was supported in part by the Hong Kong Research Grants Council (RGC) ECS under Grant 26208723; in part by the National Natural Science Foundation of China under Grant 92364102 and Grant 62304192; and in part by the Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project) under Grant 2023A03J0013. This article was recommended by Associate Editor W. Qian. (Corresponding author: Zhiyao Xie.)

Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, SAR (e-mail: wenjifang1@ust.hk; yludf@connect.ust.hk; sliudx@connect.ust.hk; qzhangcs@connect.ust.hk; hongcezh@ust.hk; eezhiyao@ust.hk).

Ceyu Xu and Lisa Wu Wills are with the Department of Computer Science and Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: ceyu.xu@duke.edu; lisa@cs.duke.edu).

Hongce Zhang is with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, SAR, and also with the Microelectronics Thrust, Function Hub, Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511458, China.

Digital Object Identifier 10.1109/TCAD.2024.3420904

I. INTRODUCTION

IN MODERN VLSI design flows, the register-transfer level (RTL) stage is a critical point, where designers devote significant effort to crafting precise design behavior descriptions using hardware description languages (HDLs), such as Verilog, VHDL, and Chisel [1]. At this early stage, design engineers face a vast design space with maximum flexibility, allowing them to make virtually any fine-grained design decisions that will affect the ultimate quality of the ASIC design in terms of power, performance, and area (PPA). Ideally, designers should optimize their RTL designs sufficiently at this stage, since it is extremely challenging, if not impossible, to remedy low-quality RTL in downstream synthesis stages.

Despite the critical importance of optimizing RTL designs, it is very difficult to evaluate the RTL design quality, considering an RTL design is still in the format of the HDL code. In a standard VLSI design flow, designers have to go through the time-consuming subsequent synthesis or even layout stages, relying on the full-fledged commercial electronic design automation (EDA) tools to evaluate the design quality based on the netlists or layouts. For complex industrial designs, the logic synthesis could take more than one day and the layout process can easily further take several days. To make things worse, designers often need to frequently invoke synthesis tools to implement and evaluate their RTL designs, optimize the RTL code based on the results, and then repeat the evaluation process until optimization is complete. This iterative process significantly prolongs the total turnaround time and hinders the optimization of design quality, making the RTL design process extremely inefficient.

In recent years, customized machine learning (ML) methods have been increasingly proposed to provide early feedback on the design quality [2]. While most ML approaches focus on the gate-level netlists or layouts, the crucial RTL stage often receives less attention [2]. In the existing ML-based methods, gate-level netlists are usually modeled as graphs and processed with graph neural networks (GNNs) and layouts are treated as images (i.e., 2-D matrices) for analysis with convolutional neural networks (CNNs). However, since the RTL design is in the HDL code format rather than the conventional data structures, there is no established consensus on the best representation and processing method for them. Some studies [3], [4], [5], [6] focus only on the design flow tuning for specific designs to achieve better PPA outcomes, but they require model retraining for every new design. This

limitation is also seen in most RTL-stage power models [7], [8], [9], [10], [11], which do not generalize well to new designs. Besides the RTL-stage PPA modeling, which is the focus of this work, there are methods targeting the earlier architectural stages [12], [13], [14], [15], [16], [17]. However, these face even greater challenges in generalizing to unseen designs due to the absence of the detailed RTL information.

Most recently, several general cross-design ML methods [18], [19], [20], [21], [22] are proposed to predict design qualities at the RTL stage. They first convert design RTL to intermediate representations, such as the bit-level and-inverter graph (AIG) [18], [19], [20] or word-level abstract syntax tree (AST) [21], [22], and then develop the ML models to evaluate the design quality metrics. Works [18], [19] develop RTL-stage timing models based on the neural networks, but these models are limited to combinational circuits and rely on the design variations generated by an RTL generator. Work [20] classifies timing paths as critical or noncritical, and then employs the predictions to optimize the logic synthesis results. However, it does not provide concrete timing predictions [e.g., worst negative slack (WNS), total negative slack (TNS)], and lacks the modeling for power and area metrics. Two recent works [21], [22] utilize the AST representation for the RTL code and then evaluate the design PPA either based on all the register trees [21] or randomly sampled paths [22] extracted from the AST-alike representations. However, the effectiveness of these methods in predicting the qualities of new, unknown RTL designs is still constrained by several factors. First, the AST-alike representation used in these studies [21], [22] is originally just the initial data format for the HDL code. It is not an ideal data format to support ML solutions. Second, these works [21], [22] process the representations with several unreasonable operations. For instance, [21] exhibits undesirable duplications in register trees during logic counting. In [22], there is a noticeable discrepancy between the pseudo-training paths and the actual paths extracted from the target inference designs.

Furthermore, accurate presynthesis PPA modeling necessitates high-quality RTL designs, which are valuable intellectual properties (IPs) for the IC design companies. However, these designs are often unavailable for model development, resulting in a scarcity of diverse RTL designs for ML model training. This shortage highlights the need for exploring effective RTL data augmentation techniques. Recently, the use of generative AI, especially large language models (LLMs) in IC design is gaining traction. Studies have explored LLMs for generating Verilog code, employing either commercial LLMs with strategic prompt engineering [23], [24], [25] or by fine tuning open-source LLMs with the Verilog data [26], [27]. The performance of LLMs is typically assessed through the benchmarks [28], [29] based on the functional correctness of the generated RTL. However, meeting specific functional requirements is not the primary goal for the data augmentation. Our emphasis lies on generating data exhibits both diversity and close similarity to real-world RTL designs.

In this work, we propose a new RTL-stage PPA modeling framework named MasterRTL, which achieves significantly higher accuracy over prior works [21], [22] when applied to

new RTL designs. The overall workflow of MasterRTL is presented in Fig. 1. It is the first work that supports the cross-design RTL evaluation on all the major PPA qualities, including both TNS and WNS, both vector-less and vector-based power analysis results, and the gate area.¹ Furthermore, MasterRTL's PPA predictions are adaptable across different design stages and technology libraries, offering broad applicability. To address the RTL data availability challenges, we have also developed data augmentation techniques. It primarily answers the following key unsolved questions in the RTL-stage PPA modeling problem.

- 1) *Q1*: What is the most appropriate data format of the RTL design (i.e., RTL representation) that best supports the ML modeling methods?
- 2) *Q2*: Based on the RTL representation, how to capture the key patterns to estimate each design objective?
- 3) *Q3*: How to extend the postsynthesis PPA evaluation across the later layout stage and distinct technology libraries?
- 4) *Q4*: How to generate and evaluate high-quality augmented RTL designs to enhance our ML model training process?

For *Q1*, we aim to develop an ML method suitable for any given RTL design. The representation for RTL must be closely *similar* to the final gate-level netlist while also being as *general* as possible. Such a *general* representation promotes uniformity among various design types, enhancing the ML model's predictive performance on unseen new designs. To this end, MasterRTL adopts a bit-level representation similar to that used in [18], [19], and [20] for the ML-based RTL analysis, named simple operator graph (SOG). It consists of only fundamental single-bit logic operations. Compared with the AST-alike representations in works [21], [22], it better unifies different RTL design types and styles and thus enables a higher cross-design model accuracy for almost all the design objectives and ML methods.

In response to *Q2*, since the mechanisms behind ground truth PPA measurements differ significantly, instead of adopting similar input features for different tasks in prior works [21], [22], we customize different estimation methodologies for timing, power, and area separately. Specifically, among all the RTL-stage cross-design methods, our timing model is the first to explicitly capture the critical path and the corresponding delay between any pair of registers. This is enabled by our SOG representation's *consistency in register mapping* with the netlist. Our power model is also the first to integrate toggle rate information as features, thus supporting unified predictions on both the vector-based and vector-less power values. Furthermore, our cross-design power model introduces module-level evaluation, a novel strategy that substantially increases the volume of power labels for model training.

For *Q3*, we enhance MasterRTL's initial postsynthesis PPA predictions by constructing transfer models that incorporate

¹In comparison, [21] only evaluates TNS and vector-less power, [22] only evaluates WNS, vector-less power, and area. Most other RTL-stage models [3], [4], [7], [8], [9], [10], [11] are not cross-design, requiring retraining on new designs.

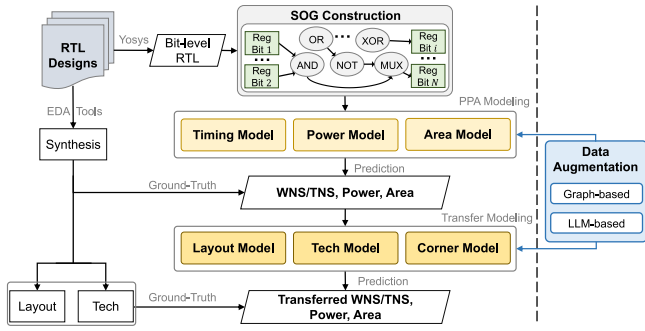


Fig. 1. MasterRTL overall workflow for RTL-stage design PPA prediction with data augmentation techniques.

design scale information. This approach enables accurate PPA modeling for the layout stage and improve transferability across various technology nodes and process corners.

Regarding Q4, we first introduce criteria for the quality evaluation of the generated RTL code, and then propose a graph-based and an LLM-based data augmentation strategy, which leverage a graph generative model and an LLM, respectively. We demonstrate the utility of the generated data in enhancing the model training process, particularly when the training data are limited.

Our contributions in this work are summarized below.

- 1) We propose an open-sourced new framework named MasterRTL to efficiently evaluate all the PPA values of any given design RTL.² Evaluated on our comprehensive dataset with 90 different RTL designs, it achieves 0.33, 0.22, and 0.15 higher absolute values in correlation R for TNS, WNS, and power estimations than the state-of-the-art solutions.
- 2) Based on the bit-level RTL representation, we customize algorithms for each design objective, capturing their different mechanisms. Among the cross-design RTL-stage methods, MasterRTL is the first to capture detailed critical-path information in timing modeling, and the first to integrate toggle rate and module-level information in power modeling.
- 3) We extend the MasterRTL predictions to the layout stage and across various technology libraries with significantly reduced error compared with the original postsynthesis netlist based on the source technology. We further explore the influence of different logic synthesis options.
- 4) We pioneer in exploring two data augmentation techniques to generate numerous new RTL designs. Experiments demonstrate their capability in mitigating the circuit data availability problem.

II. METHODOLOGY

This section delves into our MasterRTL framework. We first denote the HDL-code format of an RTL design \mathcal{H} , and the postsynthesis gate-level netlist as \mathcal{G} , with its timing, power, and area as $\{T_{\mathcal{G}}, P_{\mathcal{G}}, A_{\mathcal{G}}\}$. The goal of our RTL modeling framework F is to evaluate these qualities of any RTL design after logic synthesis. MasterRTL begins by transforming the

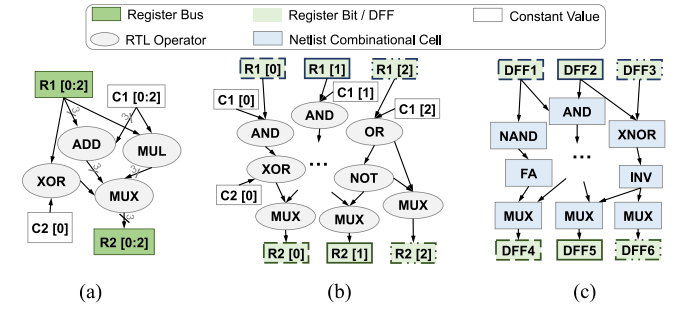


Fig. 2. Comparison between different RTL representations and the target gate-level netlist. (a) AST-alike (b) SOG. (c) Netlist.

HDL-code \mathcal{H} to a representation \mathcal{R} , facilitating the detailed analysis of RTL. Subsequently, distinct timing, power, and area models $\{f_t, f_p, f_a\}$ will be developed independently. The target can be expressed as follows:

$$F(\mathcal{H}) = \{f_t(\mathcal{R}), f_p(\mathcal{R}), f_a(\mathcal{R})\} \rightarrow \{T_{\mathcal{G}}, P_{\mathcal{G}}, A_{\mathcal{G}}\}. \quad (1)$$

Then, we further apply the transfer model Tr to the predicted PPA data to evaluate the new metrics denoted as $\{T_{\mathcal{G}'}, P_{\mathcal{G}'}, A_{\mathcal{G}'}\}$, shown as the formula below

$$\text{Tr}(F(\mathcal{H})) \rightarrow \{T_{\mathcal{G}'}, P_{\mathcal{G}'}, A_{\mathcal{G}'}\}. \quad (2)$$

In this section, we will first illustrate the novel RTL representation \mathcal{R} named SOG adopted by MasterRTL. Based on this new representation \mathcal{R} , we will delve into the new timing, power, and area models proposed in MasterRTL. Then, we will illustrate how the MasterRTL's modeling approach can be extended to the layout stage and adapted for various technologies. Finally, we will outline our data augmentation strategies.

A. SOG: Our Suggested Bit-Level RTL Representation

The RTL-stage PPA modeling begins with transforming the raw HDL code, denoted as \mathcal{H} , into a structured design format, represented as \mathcal{R} . This transformation is crucial for interpreting the detailed RTL design data. The primary challenge lies in linking the RTL-stage representation to the postsynthesis gate-level netlist without relying on the time-consuming logic synthesis process. Prior works [21], [22] have shown a direct conversion of the HDL code into a word-level AST-alike format, shown in Fig. 2(a). Fig. 2(c) demonstrates the postsynthesis gate-level netlist of this RTL design.

Rather than the word-level AST, MasterRTL follows [18], [19], and [20] to employ the bit-level RTL representation that bypasses the logic optimization and technology mapping in standard synthesis procedures. As illustrated in Fig. 2(b), this bit-level representation, \mathcal{R} contains only single-bit registers and five primary single-bit logic operations: 1) two-input AND, 2) OR, 3) XOR, 4) NOT, and 5) a two-to-one MUX, and we name it as SOG. The process of creating the SOG involves breaking down each multibit word into its logic bits and replacing complex word-level operations with their Boolean equivalents, following a preset mapping relationship. This SOG generation is achieved through the open-source tools like Yosys [30], ensuring a swift and efficient process.

²It has been open-sourced in <https://github.com/hkust-zhiyao/MasterRTL>.

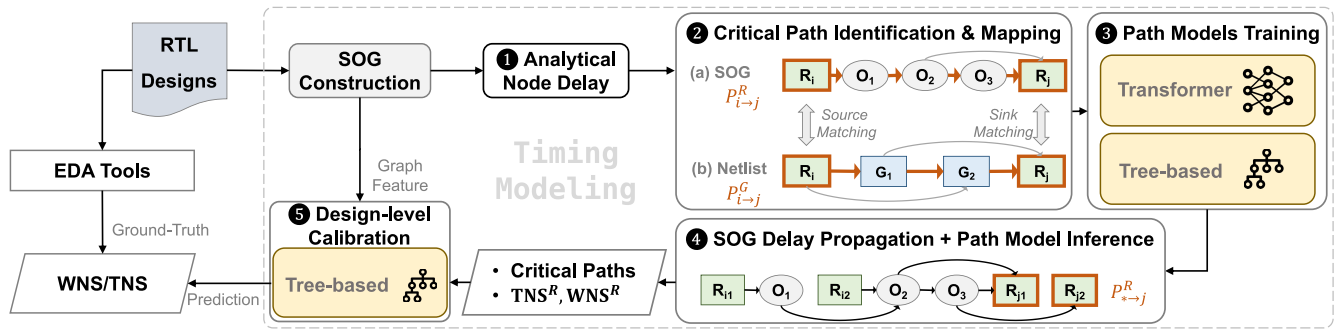


Fig. 3. Timing evaluation flow in MasterRTL. The multistage timing model identifies critical paths at the RTL stage enabling accurate WNS and TNS evaluations.

We summarize the key features and advantages of adopting bit-level SOG, in comparison to the AST-alike representation used in [21] and [22], reveals three key advantages.³

- 1) *Similarity*: SOG's structure more closely mirrors that of the target netlist after synthesis, effectively bridging the gap between the presynthesis RTL and postsynthesis gate-level netlist.
- 2) *Generalization*: SOG relies on only five fundamental single-bit logic operations, offering broader generalization capabilities than the AST. This simplification reduces variations across different RTL designs and styles.
- 3) *Consistency in Register Mapping*: Each register cell in the gate-level netlist \mathcal{G} has a corresponding single-bit register operator in SOG \mathcal{R} . This direct mapping is a significant improvement over the AST's word-level nodes, which fail to directly map to the netlists.

Benefiting from the consistent one-to-one register mapping of the bit-level representation, we customize the ML models based on the distinct PPA mechanisms for the accurate RTL-stage PPA evaluation. Specifically, our timing modeling identifies the critical paths in the SOG, which are then mapped onto the netlist, aligning with the paths that have identical starting and ending registers. Our power modeling method propagates toggle rates from the registers to the logic gates, a crucial factor for dynamic power modeling. Additionally, our area model utilizes key features, such as calculated area from the SOG, significantly enhancing the prediction accuracy. In the subsequent sections, we will delve into the specifics of our modeling methods and RTL data augmentation techniques.

B. RTL-Stage Timing Modeling

For the RTL-stage timing modeling, as depicted in Fig. 3, we introduce a multistage ML framework to evaluate both the TNS and WNS of any RTL design. The target ground truth TNS and WNS are obtained from the postsynthesis timing report. Notice that, detailed timing evaluation at such an early stage is extremely challenging, since the logic optimization and technology mapping have not been carried out. Therefore, different from the timing modeling methods applied at later stages (i.e., the netlist-stage or the layout-stage) [32], [33], [34], our modeling method will primarily

focus on the key patterns and use some approximations. Despite the approximations inherently introducing imperfections, we mitigate these by subsequent calibration exploiting the ML models.

A primary challenge in RTL-stage timing modeling is that the ground truth labels are derived from the timing analysis of the gate-level netlist \mathcal{G} . Such netlist-level timing values cannot be directly mapped to the AST-alike RTL representation \mathcal{R} . As a result, state-of-the-art studies either overlook the RTL details [21] or turn to use synthesized pseudo paths for training [22], which often leads to significant inaccuracies. In comparison, the MasterRTL framework employs the consistency of registers between the SOG representation \mathcal{R} and the netlist \mathcal{G} , offering a more accurate approach to timing modeling at the RTL stage. Specifically, we will capture the slowest critical paths in both the SOG representation \mathcal{R} and netlist \mathcal{G} , and map these paths one by one according to their starting and ending registers. This multistage timing modeling process is introduced in detail below.

1 Node-Delay Modeling in \mathcal{R} : To facilitate the assessment of delays at each node of our RTL representation \mathcal{R} in SOG, we develop a simplified analytical (non-ML-based) model. Note that the “node-delay” calculated for \mathcal{R} does not represent an actual delay value. Instead, its purpose is to assist in path extraction on \mathcal{R} for feature collection. This node-delay model operates as a linear function of the fan-out number, whose coefficients are based on the type of the driving node operator. These coefficients for each node operator type are approximated using the resistance-capacitance (RC) values of the standard cells corresponding to the same type, as found in the liberty files (e.g., lib/db).

2 Critical Path Identification and Mapping in \mathcal{R} and \mathcal{G} : Leveraging the estimated node delay in \mathcal{R} , we then identify the maximum-delay path $P_{i \rightarrow j}^{\mathcal{R}}$ between the two registers (i.e., the start register i and the end register j) in representation \mathcal{R} , where $i, j \in [1, N]$ in a design with N registers. This is implemented by efficiently propagating the node delay across the SOG graph in a topological order.

Following the identification of the critical path $P^{\mathcal{R}} i \rightarrow j$ in RTL representation \mathcal{R} , our aim shifts to predict the actual maximum path delay between the same register pair in the gate-level netlist, \mathcal{G} . This targeted path, $P^{\mathcal{G}} i \rightarrow j$ is between the registers i and j . We collect its ground truth path delay label using the postsynthesis timing analysis EDA tools.

³Refer to our previous published version [31] for more detailed evaluations.

It is important to note that while the critical paths $P^{\mathcal{R}}i \rightarrow j$ and $P^{\mathcal{G}}i \rightarrow j$ in representations \mathcal{R} and \mathcal{G} share the same start and end registers i, j , the actual nodes comprising each path differ significantly. As illustrated in Fig. 3, a node on $P^{\mathcal{R}}i \rightarrow j$ corresponds to an RTL operator, whereas on $P^{\mathcal{G}}i \rightarrow j$, it represents a standard cell.

③ *Path-Level Delay Model Training*: We then develop a path-level model, f_t^{path} trained using features from the RTL-stage path $P^{\mathcal{R}}i \rightarrow j$, to predict the actual path delay of the corresponding netlist path $P^{\mathcal{G}}i \rightarrow j$. The model's function can be expressed as

$$f_t^{\text{path}}\left(P_{i \rightarrow j}^{\mathcal{R}}\right) \rightarrow \text{The path delay of } P_{i \rightarrow j}^{\mathcal{G}}. \quad (3)$$

For the training data, we focus on the register pairs i, j that form the top 1% of maximum-delay paths in the netlist $P_{i \rightarrow j}^{\mathcal{G}}$, as identified in the postsynthesis timing reports for each design.

For the proposed path-level model f_t^{path} , we investigate two ML approaches. First, we employ a transformer model, adopted in LLMs, to process each path as a sequence of nodes. This model is similar to the one adopted in [22]. However, to address the limitations of [22], which lacks fan-out information in its features, a key element for the accurate path delay prediction, we incorporated the fan-out count of each node in $P_{i \rightarrow j}^{\mathcal{R}}$ into our input features. Second, we explore a lightweight tree-based model, such as random forest [35], with careful feature engineering. The extracted features from $P_{i \rightarrow j}^{\mathcal{R}}$ include: 1) the total count of all nodes; 2) the quantity of each operation type; and 3) the accumulated node delay on this path, as defined in our earlier model in ①.

④ *Path-Level Delay Model Inference*: Once the path-delay model f_t^{path} is trained, it is employed for TNS and WNS prediction in new RTL designs. This process in representation \mathcal{R} mirrors the TNS and WNS calculation methods used in the netlists. For each register, identified as the endpoint, we capture its critical path. Employing the technique from ②, we propagate the estimated node delays through the SOG graph. This approach results in the identification of N unique paths, each denoted as $P_{* \rightarrow j}^{\mathcal{R}}$ for $j \in [1, N]$. In this context, $*$ signifies the starting register that culminates in the maximum path delay to each respective endpoint register j .

Following the identification of the N paths $P_{* \rightarrow j}^{\mathcal{R}}$, the trained path-level model f_t^{path} is employed to predict the delay for each of these paths. Based on these predictions, we calculate the TNS and WNS estimations within the representation \mathcal{R} as follows:

$$\begin{aligned} \text{TNS}^{\mathcal{R}} &= \sum_{j=1}^N \left(\text{clk} - f_t^{\text{path}}\left(P_{* \rightarrow j}^{\mathcal{R}}\right) \right) \\ \text{WNS}^{\mathcal{R}} &= \min_{j \in [1, N]} \left(\text{clk} - f_t^{\text{path}}\left(P_{* \rightarrow j}^{\mathcal{R}}\right) \right). \end{aligned}$$

⑤ *Design-Level TNS/WNS Calibration*: Given that the RTL-stage timing modeling is highly challenging, the estimated path delays, $\text{WNS}^{\mathcal{R}}$ and $\text{TNS}^{\mathcal{R}}$ predicted directly by the path-level model in ④ are not sufficiently accurate. However, they serve as valuable starting points for further refinement and

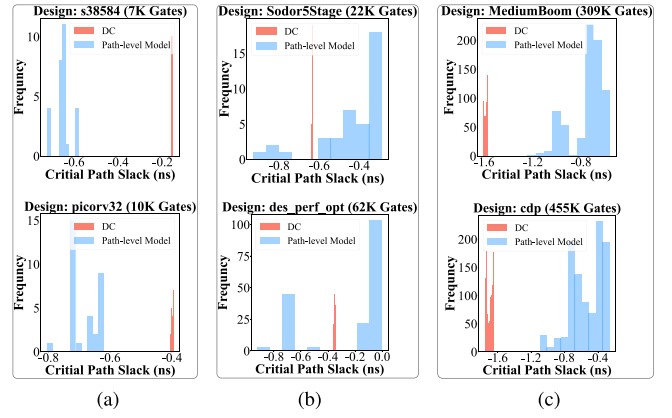


Fig. 4. Delay distribution of the worst 1% critical paths from synthesized netlist (DC) and our path-level model. (a) Small designs. (b) Medium designs. (c) Large designs.

calibration. In Fig. 4, we present a comparative analysis of the slack distribution for the worst 1% of the N critical paths, as predicted by our path-level model ($\text{clk} - f_t^{\text{path}}(P_{* \rightarrow j}^{\mathcal{R}})$) versus the actual slack obtained from the ground truth netlist timing report. We observe several interesting patterns in Fig. 4.

- 1) The distribution of critical paths obtained from the netlists is notably more concentrated compared to the predictions made by the path-level model.
- 2) Despite the huge discrepancies between the ground truth and the predictions made by the path-level model, consistent patterns emerge when considering the size of the designs. Specifically, predictions for the small-scale designs tend to be overly pessimistic, large-scale ones are over-optimistic, and medium-sized designs fall in the middle.

The observed discrepancy patterns are expected and can primarily be attributed to the optimization processes conducted during the logic synthesis. These optimizations are specifically directed toward the most critical paths. As a result, in the netlist \mathcal{G} , the slacks of the top 1% critical paths become more concentrated and closely align with the actual WNS. Furthermore, the influence of these optimization efforts is more obvious in smaller designs with fewer paths. Consequently, this leads to the path-level predictions based on the representation \mathcal{R} being relatively over-pessimistic and vice versa.

Based on these valuable patterns, we introduce an additional final-stage model aimed at calibrating the estimated $\text{TNS}^{\mathcal{R}}$ and $\text{WNS}^{\mathcal{R}}$ values toward the actual TNS and WNS labels from the gate-level netlist \mathcal{G} . This model utilizes a tree-based ML approach and incorporates the following features: 1) the SOG features (i.e., counts of node operators) reflecting the design scale; 2) the estimated $\text{TNS}^{\mathcal{R}}$ and $\text{WNS}^{\mathcal{R}}$ from the path-level model; and 3) the slack distribution of the worst 1% of the N critical paths based on the path-level model prediction $\text{clk} - f_t^{\text{path}}(P_{* \rightarrow j}^{\mathcal{R}})$. For each design, we extract the worst, 10%, 50%, and 90% percentiles of these predicted slacks as features.

C. RTL-Stage Power Modeling

For RTL-stage power modeling, we introduce a novel toggle-rate-based and module-level method as shown in Fig. 5.

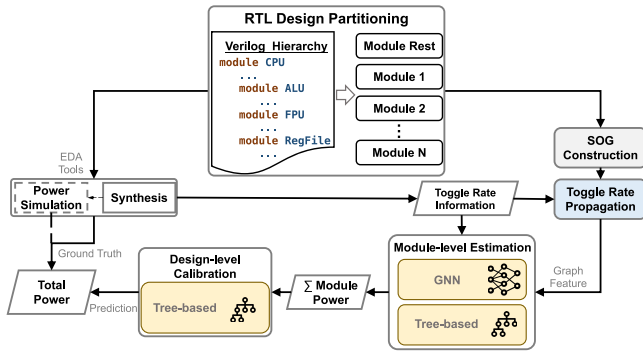


Fig. 5. Power evaluation flow in MasterRTL. The multistage power model utilizes the toggle rate information propagated from each RTL module facilitating precise power evaluation.

Distinct from the previous works [21], [22], this is the first cross-design RTL-stage power model that incorporates toggle rates as a primary input. In our methodology, each node within the SOG representation \mathcal{R} is annotated with a toggle rate, a feature that is versatile enough to support both the vector-based and vector-less power simulation scenarios. For the vector-based power analysis, the RTL simulations are conducted to gather precise toggle rate data. This approach effectively captures the dynamic behavior of the design, leading to an accurate estimation of the toggle rates, and consequently more precise power analysis. For the vector-less power analysis, the toggle rates are derived at the beginning of the logic synthesis process using the synthesis tools. This strategy provides essential insights into the design’s toggling behavior without the need for explicit workload simulation.

Node Toggle Rate Propagation: The toggle rate for all the register bits can be sourced from the switching activity interchange format (SAIF) files generated by the EDA tools. This method is applicable for both the vector-based and vector-less analysis scenarios through the workload simulation and logic synthesis, respectively. Then, the toggle rates are directly mapped onto the register nodes within the SOG representation. The key process involves efficiently propagating these toggle rates to every node in the SOG graph \mathcal{R} . This is achieved by traversing the graph in topological order and calculating based on the functionality [36] of each simple operator type within SOG.

Module-Level Power Estimation: A primary obstacle in design power prediction is the shortage of the training data, as each design contributes only one data sample (i.e., its total power). To address this, our approach does not aim to predict the total power of an entire design directly. Instead, we decompose the design into M parts, based on the RTL module hierarchy of the HDL code.⁴ This module-level method significantly enriches the total amount of the power data by M times. We estimate the power of each individual module-level partition, denoted as $\text{Power}^{S1}, \text{Power}^{S2}, \dots, \text{Power}^{SM}$. The overall power of the design, Power^S , is then calculated by summing the power of all the modules: $\text{Power}^S = \sum_{i=1}^M k_i \cdot$

⁴In this work, we focus on the partitioning modules one level below the top module.

Power^{S_i} , where $k_i \in \mathbb{Z}^+$ represents the number of times the i th module is instantiated.

In our module-level power prediction, two types of models are explored: 1) a GNN model and 2) a lightweight tree-based model. The estimation of total power incorporates both the dynamic and static power components, in which the dynamic power is particularly sensitive to the toggle rates. For the GNN model, we utilize the sub-SOG converted from the modules as the input of the model. The features for each node include: 1) number of fan-in and fan-out; 2) one-hot encoding of the six operator type; and 3) propagated toggle rate. In the tree-based approach, feature engineering is performed for each module, primarily based on the toggle rate information. The features include: 1) the sum of the toggle rate; 2) average toggle rates; 3) the sum of fan-out number multiples toggle rate on each node; and 4) the total number of nodes. For static power, which is independent of the toggle rate and typically forms a minor part of the total power, only SOG-related features are utilized.

Design-Level Power Calibration: We further add a final-stage tree-based ML model to refine power predictions, similar to the calibration in timing. This model calibrates the total power estimation, derived from the sum of the power predictions across all the modules, and integrates SOG graph features that represent the design scale. This step enhances the accuracy of our power model.

D. RTL-Stage Area Modeling

Area modeling is more straightforward than timing and power modeling, as evidenced by our experimental observations. By leveraging the SOG representation, a simple one-stage tree-based model achieves accurate area predictions. The total gate area is categorized into sequential and combinational components. The prediction of the sequential area is streamlined by calculating the product of the total register count in the SOG and the area of a standard D-flip-flop cell, as defined in the liberty file. For the combinational area, initial calculations are performed for all the operators within the SOG based on the liberty file. These preliminary values, along with the SOG-derived features are then processed by a lightweight tree-based model to estimate the combinational area.

E. Transferring Across Design Stage and Technology

MasterRTL’s original predictions are based on the postsynthesis PPA labels for a specific technology. However, in real VLSI design scenarios, both the layout stage with the physical information and varying technologies significantly impact the final PPA outcomes. Here, we further propose transfer methods to evaluate the PPA metrics at the layout stage and across different technology libraries based on the original predictions.

Regarding extending to layout PPA modeling, we observe that the disparities between the netlist-stage and layout-stage primarily arise from the offsets, while maintaining a high correlation. Moreover, the physical-design EDA tools significantly optimize the timing slacks but at an increased power consumption cost with negligible area changes. The results will be further elaborated in Section III-B.

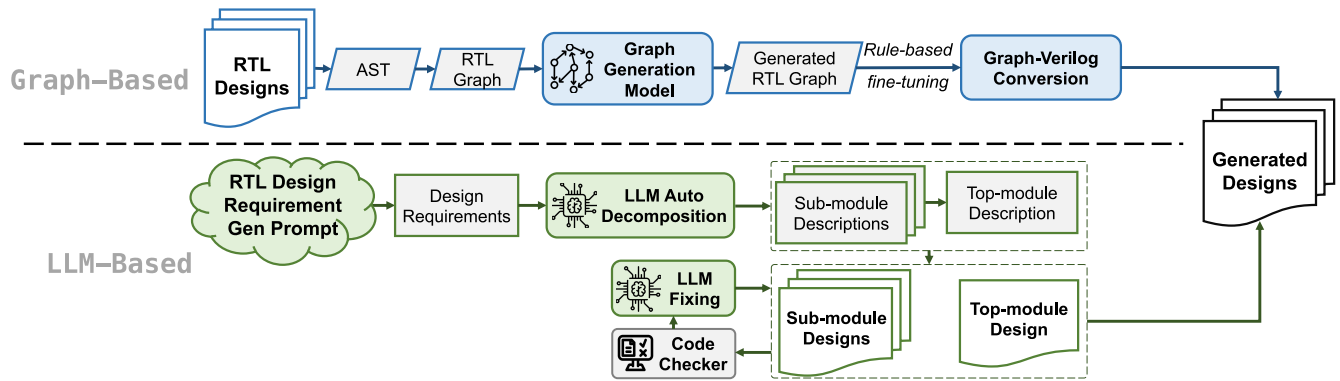


Fig. 6. Two Data augmentation strategies in MasterRTL. The graph-based method utilizes the structural information of RTL designs to generate new RTL graphs. The LLM-based method generates hierarchical RTL codes with various functionalities.

Based on the observations, we developed a transfer model to predict the postplacement PPA metrics at the RTL stage, a notably challenging task. Instead of directly targeting the post-placement PPA, our model evaluates the variations between the postsynthesis and postplacement metrics. It utilizes both the MasterRTL’s original predictions and design scale-related features (e.g., numbers of different cells) as the model input.

In terms of the technology transfer, our approach encompasses not only different technology nodes but also distinct process corners within each technology. Similar to the layout-stage extension, we evaluate the differences between the target technology libraries or process corners, relative to the source technology. The selected features include: 1) initial PPA predictions for the source technology; 2) scale calculations between the target and source libraries from the library data; 3) scaled PPA metrics; and 4) design scale-related features.

F. Data Augmentation by Generating New RTL Designs

To mitigate the shortage of high-quality RTL designs, we propose data augmentation methods to enhance the model training with the synthetic RTL data. Ideally, the augmented RTL data should be as similar to the real RTL designs as possible. We first propose three criteria to comprehensively evaluate the quality of the generated designs.

- 1) *Synthesizability*: The generated design should be fully synthesizable to ensure the acquisition of corresponding design PPA labels.
- 2) *Diversity*: To ensure a well-rounded training set, the generated designs need to encompass a broad spectrum of design categories and scales.
- 3) *Hierarchical Structure*: Reflecting the standard RTL design process, it is advisable that the synthetic designs include decomposed submodules that are separately designed and then instantiated in a top module, facilitating the PPA optimization.

Following the established criteria, we introduce two novel strategies to generate new RTL data: 1) a graph-based method and 2) an LLM-based approach.

Graph-Based Data Augmentation: Our first strategy leverages the structural characteristics of the RTL designs. It employs a graph generative method coupled by a rule-based

fine-tuning algorithm, as illustrated in the graph-based part of Fig. 6.

Specifically, we first convert a small number of available training RTL designs to the RTL representation⁵ in the graph format. In the RTL graph, nodes represent registers and operators, while edges depict their connection relationships. Then, we adopt the graph generative model in [37] to generate undirected graphs learning similar structure in our training RTL designs. This model utilizes a GNN with attention mechanisms to establish connections between our existing graph structure and the newly generated graph. It sequentially constructs clusters of nodes and their corresponding edges. Given that the generated edges are undirected, we further incorporate a classification model to predict the direction of these edges. This is achieved by using the pairwise embeddings of the two nodes connected by an edge as input features for a binary classifier. The combination of these two models enables us to accurately generate graph structures effectively reflecting the real-world RTL designs.

After that, we establish specific RTL graph rules, which are essential for ensuring the accurate conversion of the graph back into the RTL code, guiding the generated graph to align with the proposed criteria. This rule-based fine tuning algorithm modifies the graph generated from the ML model to more closely resemble the real-world RTL designs using the following predefined rules as follows.

- 1) *Operator Proportions*: We adjust the proportions among different types of operators. According to our observation, logic operators like AND, NOT, MUX dominate, while complex arithmetic operators, such as adders and multiplexers are less frequent.
- 2) *Operator Width Calculation*: We ensure the operator widths are legal and similar to the real designs.
- 3) *Redundant Cycle Elimination*: We remove any self-loops in the combinational logic to prevent cycles.
- 4) *Fan-In/Fan-Out Reduction*: We split nodes with high degrees to reduce their in-degree and out-degree. Using

⁵For the RTL generation task, we employ the AST-like graph to train the graph generative model. As previously discussed, the SOG closely resembles the gate-level netlist, whereas the AST-like representation is more similar to the original RTL.

these rules, we traverse the generated graph in topological order and assign attributes to each node accordingly. Under this process, we can generate unlimited RTL codes by relying solely on a little training data. Finally, the fine tuned graphs are converted back to the RTL code to serve as augmented designs for model training.

LLM-Based Data Augmentation: Although the graph-based solution can generate varying scales of synthesizable Verilog code, its capacity to control the functionality and hierarchy of the target RTL is still limited. Recently, the LLMs have gained significant traction in the field of the RTL code generation. However, the current RTL generation tasks are constrained by specific design descriptions and tend to be suited only for the small-scale designs. To address these limitations, we introduce an innovative approach: instead of directly generating new RTL code, we propose a technique named auto-decomposition. This method is designed to enhance both the hierarchy and scalability of the generated code. The workflow of our LLM-based method is illustrated in Fig. 6.

In our LLM-based approach, the process begins with the user specifying design categories. Based on these categories, the LLM generates a set of specific design requirements. Then, the LLM is prompted to decompose these requirements into multiple submodules, each subsequently implemented in the Verilog code. This decomposition is key to creating a hierarchical structure in the RTL design, allowing for more complex and scalable designs. The next phase involves the LLM creating a top module that effectively instantiates and interconnects all the submodules. We employ the logic synthesis tool to check the synthesizability of the generated designs. If any syntax errors are detected during synthesis, the specific code lines and error report are fed back into the LLM. The LLM then refines the generated design accordingly, iterating until the design meets all the synthesizable requirements.

Currently, these two RTL data generation methods are helpful in situations when actual RTL data is limited. The experimental evaluation of the augmented data benchmarks is detailed in Section III-E. With the ongoing trend of scaling up ML models, the lack of sufficient data will become a significant obstacle in RTL modeling. Therefore, we consider this RTL generation approach to be exceptionally promising, as it is capable of generating a nearly limitless number of new RTL designs.

III. EXPERIMENTAL RESULT

A. Experimental Setup and MasterRTL Implementation

In this work, a comprehensive dataset is created by collecting 90 different open-source RTL designs from various benchmark sources. This extensive dataset facilitates a thorough evaluation of the proposed methodology.

Table I outlines the diverse sources of the RTL designs used in the dataset. These designs are coded in various mainstream HDLs, including Verilog, VHDL, Chisel, and SpinalHDL. The dataset covers a broad spectrum of functionality, including CPU cores, vector arithmetic, ML accelerators, cryptographic units, and other designs for logic synthesis studies.

TABLE I
RTL DESIGNS FOR DATASET PREPARATION

Source Benchmark	#. of Designs	Original HDL Type	Design Scale (#K Gates) {Min, Median, Max}
ISCAS'89 [38] [†]	5	Verilog	{1, 6, 7}
ITC'99 [39] [†]	13	VHDL	{4, 10, 45}
OpenCores [40] [†]	15	Verilog	{2, 7, 62}
VexRiscv [41]	26	SpinalHDL	{7, 132, 530}
RISC-V Cores*	5	Verilog	{7, 10, 17}
NVDLA [42]	8	Verilog	{6, 40, 672}
Chipyard [43] [‡]	18	Chisel	{1, 25, 921}

For each design, the RTL description is synthesized with Synopsys Design Compiler 2021, and the physical design is conducted with Cadence Innovus 2022. The NanGate 45 nm technology library [44] with typical process corner is utilized for the original result, while the other two process corners (i.e., maximum and minimum) and four TSMC technology libraries (i.e., 22, 28, 40, and 65 nm) are used for transfer modeling. The PPA values of the gate-level netlist are evaluated using Synopsys Prime Time 2021, and are recorded as the ground truth label. The generation of SOG first begins with the generic synthesis process from the existing logic synthesis tools (i.e., Yosys [30]). It involves reading the original RTL designs and performing technology-independent transformations that convert the word-level RTL into a bit-level RTL representation. Subsequently, the bit-level RTL is parsed using a Verilog parser [45] and transformed into a graph data structure, following a similar approach as in [21] and [22]. All the experiments are conducted on a server equipped with a 2.9 GHz Intel Xeon Platinum 8375C CPU and 256 GB RAM.

We implemented and evaluated the proposed ML models using the constructed dataset. A ten-fold cross-validation method was adopted to ensure accuracy in the model assessments. Hyperparameter tuning for each ML model was conducted using a separate validation set. The final configurations and parameters of the models, after exploration and tuning are detailed in Table II.

We use four metrics to evaluate the prediction accuracy between the predicted value \hat{y} and ground truth y of $n = 90$ designs. They are: correlation coefficient (R), mean absolute percentage error (MAPE), mean absolute error (MAE), and root relative square error (RRSE) as defined below

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{y_i} \times 100\% \quad (4)$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (5)$$

$$\text{RRSE} = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (6)$$

The \bar{y} represents the average of the measured values. These metrics bring a comprehensive and fair evaluation of the ML models from different aspects, where the higher correlation and lower error indicate better accuracy. Specifically, R measures a linear correlation between the predicted values and the

TABLE II
DETAILED ML MODEL IMPLEMENTATION, INCLUDING ALL MODELING TARGETS WITH EACH INTERNAL LEVEL OR METHOD

Method	ML Model	Description / Hyperparameters
Timing	Path-level	Transformer
		Random Forest Regressor
	Design-level	XGBoost Regressor
Power	Module-level	Graph Neural Network
		XGBoost Regressor
	Design-level	XGBoost Regressor
Area	Design-level	XGBoost Regressor
Transfer	Layout & Tech	XGBoost Regressor
Generation	Graph-based	Graph Recurrent Attention Network, with 7 layers (hidden dim: 256, embedding dim: 256), 1 block size, 1 sample stride, Adam optimizer, 0.0001 learning rate, 50 training epochs
	LLM-Based	Generative Pre-trained Transformer
		GPT-4 from OpenAI

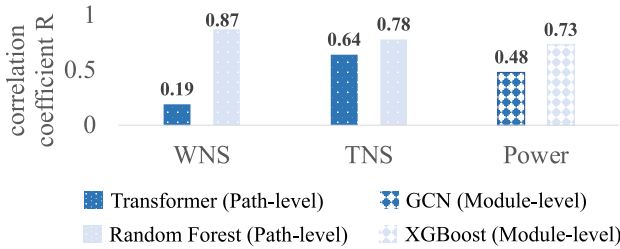


Fig. 7. Evaluation of intermediate-stage ML models in MasterRTL. The lightweight tree-based models outperform the deep learning methods (i.e., Transformer and GCN).

ground truth values, MAPE gives an intuitive percentage of disparity, while MAE directly indicates the absolute difference. RRSE can represent the accuracy ignoring the influence of different RTL design scales. Here, the MAE is employed only when the label values are exceedingly small or zero, such as in the transfer model evaluations, and MAPE and RRSE are utilized in other contexts.

In presenting our experimental results, we focus on comparing our approach with the originally introduced methods in [21] and [22], which serve as our primary baselines. Additionally, we have implemented a variety of baseline variations to conduct thorough the ablation studies. We have endeavored to optimize performance across all the configurations and variations via a consistent model-tuning process.

B. PPA Estimation Accuracy Evaluation and Comparison

Since, our PPA modeling methods contain multiple levels and include decomposed components for power and area, we first evaluate the prediction accuracy at the internal levels and individual components. Subsequently, we will delve into the accuracy of the final PPA prediction of MasterRTL.

Fig. 7 compares the explored intermediate-stage ML models in MasterRTL. At the path-level for timing models, the random forest model, which utilizes manually extracted path features, outperforms the Transformer model. A similar trend is observed in module-level power models, where the XGBoost regressor shows greater accuracy compared to the GCN model. Consequently, the light-weight tree-based models are preferred in the intermediate stages of MasterRTL, instead of the deep learning-based models.

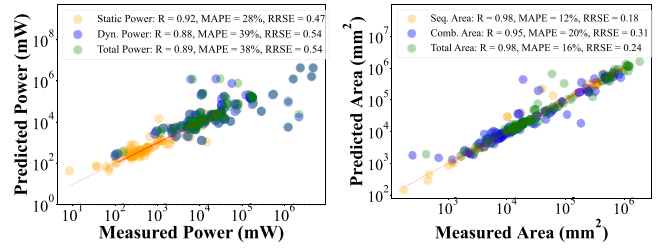


Fig. 8. Accuracy evaluation for individual components of power and area modeling.

In addition, the breakdown components of total power and total area are illustrated in Fig. 8. Total power comprises static and dynamic power. Our predictions show that the static power being independent of the toggle rates and a minor contributor to total power, aligns closely with the ground truth. In contrast, dynamic power, which is more challenging to assess results in a less accurate total power prediction. Regarding area prediction, the total area is the sum of the sequential and combinational areas. Sequential area, corresponding to registers in the SOG, can be accurately calculated based on the number of the registers due to their one-to-one relationship with the target netlist. The combinational area, represented and modeled through our SOG approach, also correlates well leading to an accurate estimation of the total area.

Table III and Fig. 9 compare MasterRTL with the previous studies [21], [22], emphasizing the difference in their approaches. As discussed, the previous works used AST-like representations, while MasterRTL utilizes SOG. The study by [21] focused on the TNS and power predictions, and [22] centered on WNS, power, and area. The table specifically highlights actual proposed methods in bold and colored cells, with other entries for the ablation studies.

We have multiple interesting observations in Table III. It reveals that the MasterRTL outperforms prior works in all the estimations, showing higher correlation coefficients (R) and lower MAPE and RRSE errors in WNS, TNS, power, and area predictions. The result also indicates a general performance improvement across the methods using SOG compared to the AST-like representations, particularly evident compared with [21]. These findings validate the effectiveness of the SOG representation as an ML-friendly RTL representation, suggesting its broader adoption in future studies.

TABLE III

COMPARISON OF EVALUATION RESULTS FOR WNS, TNS, TOTAL POWER, AND TOTAL AREA. ROWS HIGHLIGHTED IN COLOR INDICATE THE INITIALLY PROPOSED METHODS. THE PRIOR WORK ICCAD'22 [21] FOCUSES SOLELY ON EVALUATING TNS AND POWER, AND ISCA'22 [22] PROPOSES TO ASSESS WNS, POWER, AND AREA

Method	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE
[21] (AST-alike)	WNS	0.37	26%	0.95	TNS	0.63	48%	0.79	Power	0.42	51%	1.01	Area	0.75	38%	0.68
[21] (SOG)		0.82	24%	0.53		0.86	41%	0.36		0.62	48%	0.79		0.94	31%	0.33
[22] (AST-alike)		0.71	35%	1.15		0.78	36%	1.1		0.74	68%	0.81		0.93	38%	0.42
[22] (SOG)		0.78	26%	0.78		0.8	29%	0.88		0.82	48%	0.62		0.96	35%	0.4
MasterRTL (AST-alike)		0.81	22%	0.6		0.95	36%	0.31		0.79	44%	0.63		0.94	31%	0.34
MasterRTL (SOG)		0.93	14%	0.4		0.96	27%	0.29		0.89	38%	0.54		0.98	16%	0.24

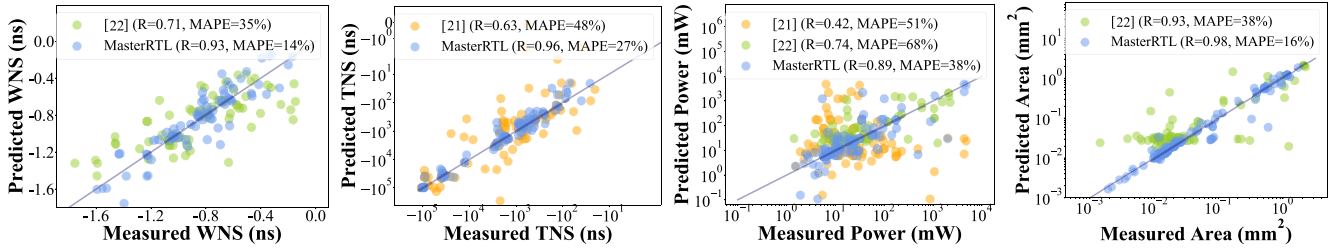


Fig. 9. Comparison between prediction and ground truth for each PPA metric across all designs. MasterRTL markedly outperforms the SOTA solutions [21], [22] in all PPA evaluations.

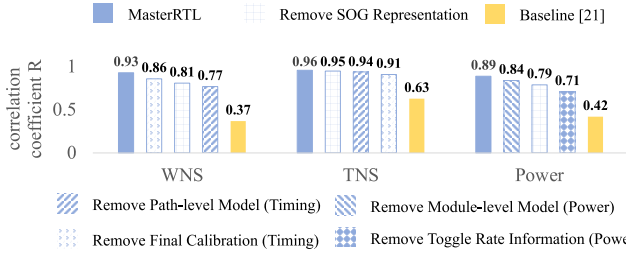


Fig. 10. Ablation study evaluating the contribution of various decomposed factors to the PPA evaluation accuracy.

Furthermore, the comparison in Table III indicates that the WNS and power estimations are generally more challenging and less accurate compared to the TNS and area, which correlate more with each design's scale. The SOG's superiority over AST is more obvious in these two challenging tasks.

C. PPA Estimation Accuracy Ablation Study

To evaluate the performance of MasterRTL, a decomposed analysis through the ablation studies is conducted by removing key policies. These crucial policies include: 1) use of SOG over AST-alike representation; 2) customized key features (e.g., predicted path slack from the path-level model for timing, module-level partition, and toggle rate propagation for power); and 3) design-level calibration.

The result of the ablation study in Fig. 10 highlights the importance of specific policies in MasterRTL. For the WNS prediction, removing the path-level model, which offers critical path information, results in the most significant accuracy drop. In the TNS prediction, the largest accuracy decline is observed when the graph features are excluded, underlining the significance of the SOG features and design scale information. The accuracy in predicting total power is notably dependent

on the toggle rate information. When all the key policies are removed, MasterRTL's accuracy aligns with the baseline method [21], demonstrating the critical role these policies play in enhancing performance.

D. MasterRTL Transfer Modeling Results

Extension to Layout Stage: In Fig. 11, we first depict the correlation in PPA between the postsynthesis labels (green) from the logic synthesis tool and the postplacement labels (the X-axis) from the physical implementation tool. Due to the lack of physical-aware optimization, the timing and power metrics after synthesis show significant MAPE gaps compared to those after placement. Our MasterRTL postsynthesis PPA predictions (blue) exhibit a similar correlation and MAPE to the postsynthesis labels (green). After extending MasterRTL to the layout stage, the postplacement PPA predictions (red) demonstrate that our framework can achieve satisfactory evaluations for the layout stage early in the RTL stage, even outperforming the postsynthesis labels.

Notably, there is a perfect correlation between the post-synthesis and postplacement area labels ($R = 1.0$ and $MAPE = 19\%$). Though the predictions made by MasterRTL place from the RTL stage shows lesser correlation, it does not require the time-consuming logic synthesis and placement processes, meanwhile still delivering a satisfactory area estimation ($R = 0.96$ and $MAPE = 24\%$).

Technology Transfer: We transfer the postsynthesis design PPA from the source technology, specifically the open-source NanGate 45 nm typical process corner to various alternatives, encompassing other process corners in NanGate and the commercial TSMC typical corner across four different technology nodes. The accuracy of this transfer is detailed in Table IV. We further visualize the four PPA metrics across technologies in Fig. 12. Our key observations are summarized below.

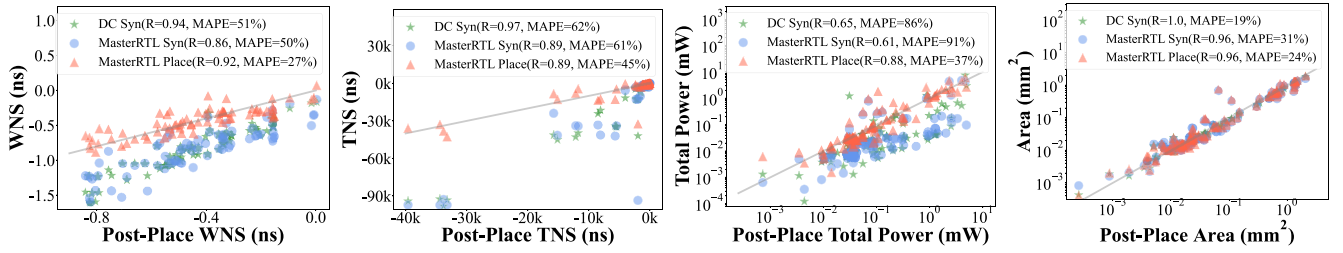


Fig. 11. Extension of MasterRTL for postplacement PPA evaluation. MasterRTL can also provide satisfactory postplacement PPA predictions at the RTL stage.

TABLE IV

EVALUATION OF THE TRANSFER MODEL ACCURACY. MAE IS UTILIZED AS THE EVALUATION METRIC. FOR EACH PPA PARAMETER, THE FIRST COLUMN (I.E., “ORI”) INDICATES THE ERROR BETWEEN THE ORIGINAL SOURCE AND TARGET LIBRARIES. THE SECOND COLUMN PRESENTS THE MAE BETWEEN OUR TRANSFERRED PPA PREDICTIONS AND THE TARGET VALUES. “IMP” SIGNIFIES THE IMPROVEMENT ACHIEVED BY OUR METHOD OVER THE SOURCE TECHNOLOGY

Source	Target	Metric	MAE			Metric	MAE (K)			Metric	MAE			Metric	MAE		
			Ori	Ours	IMP		Ori	Ours	IMP		Ori	Ours	IMP		Ori	Ours	IMP
NanGate 45nm	TSMC	WNS (ns)	4.3	2.1	51%	TNS (ns)	15.7	3.9	75%	Power (mW)	0.545	0.168	69%	Area (mm ²)	0.115	0.010	92%
			4.1	1.7	59%		15.9	3.1	80%		0.544	0.370	32%		0.113	0.011	90%
			1.7	1.6	6%		5.2	1.5	71%		0.819	0.544	33%		0.030	0.007	78%
			3.5	2.6	26%		54.5	29.8	45%		1.329	0.560	58%		0.139	0.018	87%
NanGate TYP	NanGate	MIN	0.4	0.03	93%	8.3	1.9	77%	0.245	0.031	87%	0.005	0.003	43%			
		MAX	2.3	0.12	95%	49.3	9.6	80%	0.160	0.017	89%	0.003	0.002	9%			

- 1) For the technology node transfer, our transfer model significantly reduces MAE of all the four PPA metrics, highlighting its effectiveness. Notably, the most successful transfer occurs for TSMC 40 nm (i.e., lowest MAE), indicating that similarities between the technologies lead to more successful outcomes.
- 2) Our transfer model also accurately predicts across different process corners, maintaining precision in all the PPA metrics for corner-to-corner transfers.
- 3) Regarding the labels of various technology nodes, while overall design metrics, including total power, total area, and TNS exhibit strong correlations across different technologies, the correlation for WNS is notably weaker. This variance primarily arises from the differing timing characteristics of standard cells in various libraries, which significantly impact the critical path optimization process during the logic synthesis.
- 4) Among different commercial technology nodes, those with similar features, like TSMC 22 nm and TSMC 28 nm show a good correlation.
- 5) As for the corner labels, all the four metrics exhibit perfect correlations ($R = 1$). However, timing metrics exhibit smaller errors at MIN corners, while MAX corners demonstrate reduced errors in power and area metrics.

E. Augmented RTL Design Evaluation

Statistical Evaluation: We first apply the evaluation criteria illustrated in Section II-F to assess the RTL designs generated by our two augmentation methods as detailed in Table V.

Due to the rule-based fine tuning algorithm in the graph-based method and the iteration syntax correction in the LLM-based method, all the newly generated RTL designs are fully *synthesizable*.

TABLE V
QUALITY EVALUATION OF THE GENERATED RTL DESIGNS

	Graph-Based	LLM-Based
Number of Designs	30	15
Synthesizable	✓	✓
Design Size (#K Gates) {min, mean, max}	{0.6, 32, 497}	{0.03, 0.5, 3}
Line of Code {min, mean, max}	{351, 729, 2049}	{52, 151, 233}
Functionality	N/A	Arithmetic, Algorithm, Processor, Protocol, etc.
Hierarchical	✗	✓

To evaluate the *diversity* of the augmented data, we measure the scale of the design using two metrics: 1) design size and 2) lines of code. Additionally, we examine the types of functionalities present in these designs. Our findings reveal that the graph-based method produces designs with a significantly broader range of scales compared to those generated by the LLM-based method. However, since the graph-based approach lacks control over the functionality of the designs, the LLM-based method demonstrates superior capability in generating a wider variety of functionalities.

Finally, in terms of *hierarchical structure*, the RTL designs generated via the graph-based method result in flattened structures. In contrast, the LLM-based method creates designs with the varied hierarchical levels. Since, each method on its own does not produce a perfect augmented dataset, we combine them as complementary strategies. This integration enables us to generate a more extensive and varied dataset, resulting in a total of 45 new designs.

t-SNE Visualization: To assess the similarity between the generated designs and real designs, we employed the t-SNE plot for visualization, as shown in Fig. 13. Specifically, the BERT language model is employed to convert both the real and generated Verilog designs into embedding vectors. These

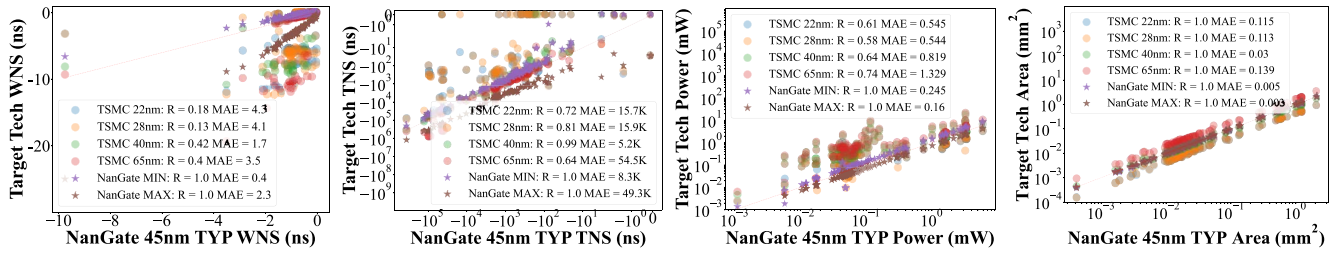


Fig. 12. Visualization of all the target technology nodes and process corners compared with the source library.

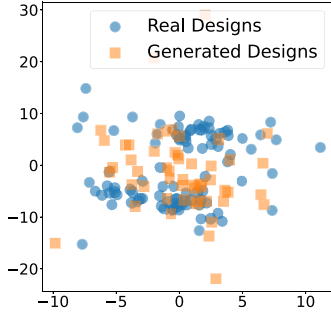


Fig. 13. t-SNE plot visualization for the similarity and diversity between the generated and real RTL designs.

embeddings are then visualized using a t-SNE plot for a thorough comparison. The plot indicates that the embedding distribution of the generated designs is similar to that of the real designs.

Experimental Evaluation: To validate the effectiveness of the generated designs, we employ them in two distinct MasterRTL modeling tasks, each with a different level of granularity. The first task involves design-level process corner transfer modeling of TNS and the second focuses on the path-level timing slack modeling. Based on the two tasks, we design two experiments to comprehensively evaluate the utility of the designs in diverse modeling scenarios within MasterRTL. The first experiment reduces the real design training data and includes all the generated designs for augmentation. The second experiment maintains the number of total training data as a constant and varies the proportions of real and generated designs. The detailed experiments are demonstrated as follows.

- 1) In our first experiment, we decrease the amount of real design training data from its full volume to nearly none, augmenting with all 45 generated designs for augmentation, 30 from the graph-based method, and 15 from the LLM-based method. Fig. 14 demonstrates the impact of varying the amount of training data, including both the design-level and timing path-level data from real designs on model accuracy. As indicated by the dashed lines, as the number of the training data from real designs diminishes, the accuracy of the model decreases sharply. Conversely, incorporating new RTL designs generated via our augmentation methods enhances the model's accuracy as shown by the solid lines. Specifically, augmentations from each source show accuracy improvements while reducing the need for the training data from real designs across all the tasks.

TABLE VI
ACCURACY OF MASTERRTL MODELING AS THE PROPORTION OF REAL DESIGNS VARIES WITH THE TOTAL NUMBER OF TRAINING DATA MAINTAINED AT A CONSTANT 45

% of real designs	MIN TNS		MAX TNS		Path slack	
	w/o Gen	w/ Gen	w/o Gen	w/ Gen	w/o Gen	w/ Gen
100%	60	64	56	74	0.88	0.9
80%	52	58	43	70	0.79	0.86
60%	43	47	33	56	0.76	0.83
40%	43	46	24	52	0.72	0.8
20%	30	44	14	43	0.63	0.77
0	/	12	/	20	/	0.64

TABLE VII
COMPARISON BETWEEN OPEN-SOURCE LOGIC SYNTHESIS TOOLS AND MASTERRTL

Method	Target	WNS	TNS	Power	Area
Yosys+ABC	R	-0.1	0.67	0.51	0.75
	MAPE	99%	95%	99%	17%
	RRSE	1	0.98	2.1	0.67
MasterRTL	R	0.93	0.96	0.89	0.98
	MAPE	14%	27%	38%	16%
	RRSE	0.4	0.29	0.54	0.24

The graph-based method is more effective than the LLM-based method, as it includes a wider range of design scales and closely resembles real design benchmarks. Additionally, combining both the augmentation sources further enhances the accuracy improvements. The improvement becomes more obvious as the amount of real design data further decreases.

- 2) Furthermore, we conduct the second experiment by keeping the total training data constant at 45. This experiment aims to evaluate the impact of varying the proportions of generated designs from 0% to 100%. The results are shown in Table VI, while reducing the proportion of real designs leads to a decline in accuracy improvement, the absence of any augmented RTL designs results in a much more significant drop in accuracy.

IV. DISCUSSION

A. Comparison With Open-Source Synthesis Tools

In addition to the ML-based PPA prediction methods for the RTL designs, open-source synthesis solutions offer fast execution and the synthesized netlist can also provide the PPA estimation. Here, we compare our MasterRTL with these open-source synthesis tools. Specifically, we employed Yosys and ABC to synthesize the RTL designs and used prime time

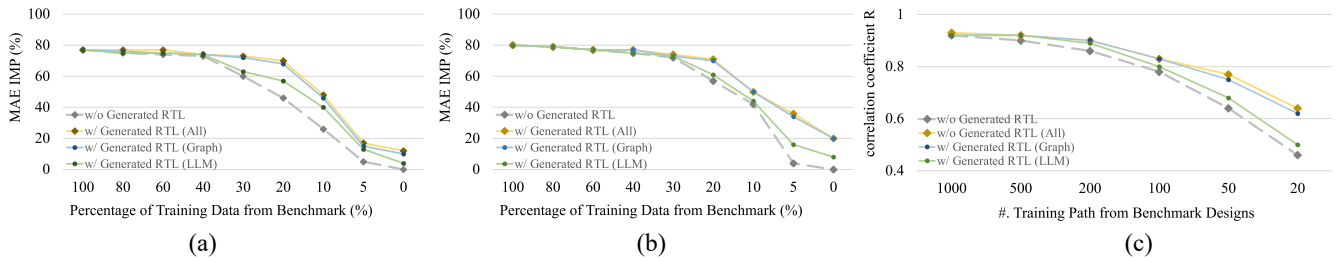


Fig. 14. Impact of reducing real design training data and introducing augmented RTL designs on model accuracy. Each source of augmentation improves accuracy across tasks with the graph-based approach outperforming the LLM-based method. Combining augmentations from both sources further boosts the accuracy. (a) TNS transfer (TYP-MIN). (b) TNS transfer (TYP-MAX). (c) Path slack prediction.

for a fair evaluation of the PPA values of the netlist, with results shown in Table VII.

The results indicate that the MasterRTL significantly outperforms open-source synthesis tools in terms of both speed and prediction accuracy for all the PPA metrics. While open-source synthesis tools are fast, MasterRTL is still 60% faster as it only performs generic synthesis, bypassing the time-consuming logic optimization and technology mapping processes. Moreover, due to significant PPA optimization by commercial tools, open-source tools struggle to correlate well with them, particularly in timing and power metrics. Area correlation is the exception, as it is determined solely by the number and types of the gate cells. However, MasterRTL still significantly outperforms in terms of area prediction accuracy.

Besides the speed and accuracy benefits, our ML-based solution also demonstrates the capability to transfer the post-synthesis results to the layout stage and across different technology libraries, a feature not available in the open-source synthesis tools.

V. CONCLUSION

In this article, we present MasterRTL, a presynthesis PPA estimation framework for RTL designs, integrated with innovative data augmentation techniques. Our method incorporates a general RTL representation named SOG and customizes the multistage ML models for WNS, TNS, power, and area separately. MasterRTL not only provides precise postsynthesis PPA evaluations but also extends to layout-stage estimations and adapts to various technology libraries. Additionally, we demonstrate two data augmentation methods to generate RTL designs resembling real-world data, thereby effectively addressing the data shortage issues.

REFERENCES

- [1] J. Bachrach et al., “Chisel: Constructing hardware in a scala embedded language,” in *Proc. Design Autom. Conf. (DAC)*, 2012, pp. 1212–1221.
- [2] M. Rapp et al., “MLCAD: A survey of research in machine learning for CAD keynote paper,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3162–3181, Oct. 2022.
- [3] Z. Xie et al., “FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning,” in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 19–25.
- [4] R. Liang et al., “FlowTuner: A multi-stage EDA flow tuner exploiting parameter knowledge transfer,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.
- [5] C. Yu and W. Zhou, “Decision making in synthesis cross technologies using LSTMs and transfer learning,” in *Proc. ACM/IEEE Workshop Mach. Learn. CAD*, 2020, pp. 55–60.
- [6] C. Yu, H. Xiao, and G. De Micheli, “Developing synthesis flows without human knowledge,” in *Proc. 55th DAC*, 2018, pp. 1–6.
- [7] Z. Xie et al., “APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors,” in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2021, pp. 1–14.
- [8] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, “PRIMAL: Power inference using machine learning,” in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
- [9] D. Kim, J. Zhao, J. Bachrach, and K. Asanović, “Simmani: Runtime power modeling for arbitrary RTL with automatic signal selection,” in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2019, pp. 1050–1062.
- [10] Z. Xie et al., “DEEP: Developing extremely efficient runtime on-chip power meters,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, pp. 1–9.
- [11] J. Yang, L. Ma, K. Zhao, Y. Cai, and T.-F. Ngai, “Early stage real-time SoC power estimation using RTL instrumentation,” in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2015, pp. 779–784.
- [12] W. R. Davis, P. Franzon, L. Francisco, B. Huggins, and R. Jain, “Fast and accurate PPA modeling with transfer learning,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–8.
- [13] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, “McPAT-Calib: A microarchitecture power modeling framework for modern CPUs,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.
- [14] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, “High-level synthesis performance prediction using GNNs: Benchmarking, modeling, and advancing,” in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 49–54.
- [15] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, “Accurate operation delay prediction for FPGA HLS using graph neural networks,” in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.
- [16] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, “Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures,” *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 97–108, 2014.
- [17] Q. Zhang et al., “PANDA: Architecture-level power evaluation by unifying analytical and machine learning solutions,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–9.
- [18] D. S. Lopera, L. Servadei, V. P. Kasi, S. Prebeck, and W. Ecker, “RTL delay prediction using neural networks,” in *Proc. IEEE Nordic Circuits Syst. Conf. (NorCAS)*, 2021, pp. 1–7.
- [19] D. S. Lopera and W. Ecker, “Applying GNNs to timing estimation at RTL: (Invited paper),” in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, pp. 1–8.
- [20] W. L. Neto, M. T. Moreira, L. Amaru, C. Yu, and P.-E. Gaillardon, “Read your circuit: Leveraging word embedding to guide logic optimization,” in *Proc. 26th Asia South Pac. Design Autom. Conf.*, 2021, pp. 530–535.
- [21] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, “How good is your Verilog RTL code? A quick answer from machine learning,” in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, pp. 1–9.
- [22] C. Xu, C. Kjellqvist, and L. W. Wills, “SNS’s not a synthesizer: A deep-learning-based synthesis predictor,” in *Proc. 49th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2022, pp. 847–859.
- [23] K. Chang et al., “ChipGPT: How far are we from natural language hardware design,” 2023, *arXiv:2305.14019*.
- [24] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, “AutoChip: Automating HDL generation using LLM feedback,” 2024, *arXiv:2311.04887*.

- [25] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *Proc. ACM/IEEE 5th Workshop Mach. Learn. CAD (MLCAD)*, 2023, pp. 1–6. [Online]. Available: <http://dx.doi.org/10.1109/MLCAD58807.2023.10299874>
- [26] M. Liu et al., "ChipNeMo: Domain-adapted LLMs for chip design," 2024, *arXiv:2311.00176*.
- [27] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "RTLCoder: Outperforming GPT-3.5 in design RTL generation with our open-source dataset and lightweight solution," 2024, *arXiv:2312.08617*.
- [28] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design RTL generation with large language model," 2023, *arXiv:2308.05345*.
- [29] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Invited Paper: VerilogEval: Evaluating large language models for Verilog code generation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–8.
- [30] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free Verilog synthesis suite," in *Proc. 21st Austrian Workshop Microelectron. (Austrochip)*, 2013, p. 97.
- [31] W. Fang et al., "MasterRTL: A pre-synthesis PPA estimation framework for any RTL design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–9.
- [32] Z. Xie et al., "Pre-placement net length and timing estimation by Customized graph neural network," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4667–4680, Nov. 2022.
- [33] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
- [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. 59th ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1207–1212.
- [35] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.
- [36] N. H. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Noida, India, Pearson Educ., 2015.
- [37] R. Liao et al., "Efficient graph generation with graph recurrent attention networks," in *Proc. 33rd Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 1–11.
- [38] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 1989, pp. 1929–1934.
- [39] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *Design Test Comput.*, vol. 17, no. 3, pp. 44–53, Jul.–Sep. 2000.
- [40] C. Albrecht, "IWLS 2005 benchmarks," in *Proc. Int. Workshop Logic Synthesis (IWLS)*, 2005, pp. 1–18. [Online]. Available: <http://www.iwls.org>
- [41] "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," VexRiscv. 2022. [Online]. Available: <https://github.com/SpinalHDL/VexRiscv>
- [42] (Nvidia Softw. Co., Santa Clara, CA, USA). *NVIDIA Deep Learning Accelerator*. (2018). [Online]. Available: <http://nvidia.org/primer.html>
- [43] A. Amid et al., "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, Jul./Aug. 2020.
- [44] "NanGate 45nm open cell library." Accessed: Apr. 7, 2011. [Online]. Available: <https://si2.org/open-cell-library/>
- [45] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for Verilog HDL," in *Proc. 11th Int. Symp., Appl. Reconfig. Comput.*, 2015, pp. 451–460. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16214-0_42



Wenji Fang (Graduate Student Member, IEEE) received the B.Eng. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2021, and the M.Phil. degree in microelectronics from the Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, in 2024, where he is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering.

His research interests include timing analysis, power modeling, and VLSI design verification.



Yao Lu received the B.E. degree from the School of Electronic Science and Engineering, Southeast University, Nanjing, China, in 2020, and the master's degree from the School of Microelectronics, Fudan University, Shanghai, China, in 2023. She is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong.

Her current research interests focus on machine learning applications in EDA.



Shang Liu received the B.E. degree in automation science and electrical engineering from Beihang University, Beijing, China, in 2023. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong.

His research interests include agile VLSI design methodologies and artificial intelligence.



Qijun Zhang received the B.Eng. degree from Tongji University, Shanghai, China, in 2022. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong.

His research interests include computer architecture and electronics design automation.



Ceyu Xu received the B.S. degree in computer engineering from the University of California at Irvine, Irvine, CA, USA, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Duke University, Durham, NC, USA.

His research interests include AI-enabled hardware design methodologies and architecture for accelerating generative models.



Lisa Wu Wills received the Ph.D. degree in computer science from Columbia University, New York, NY, USA.

She is an Assistant Professor with the Department of Computer Science and Electrical and Computer Engineering, Duke University, Durham, NC, USA. Her research interests include computer architecture, accelerators, energy-efficient computing on high-performance computing, and emerging applications related to big data, machine learning, and computational biology.

Hongce Zhang (Member, IEEE) received the B.S. degree in microelectronics from Shanghai Jiao Tong University, Shanghai, China, in 2015, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ, USA, in 2021.

He is currently an Assistant Professor with Microelectronics Thrust, Function Hub of Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, and is also affiliated with the Department of Electronic and



Computer Engineering, Hong Kong University of Science and Technology, Hong Kong. His research interests include formal verification and hardware model checking.

Zhiyao Xie (Member, IEEE) received the B.Eng. degree from the City University of Hong Kong, Hong Kong, in 2017, and the Ph.D. degree from Duke University, Durham, NC, USA, in 2022.

He is an Assistant Professor with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong. His research interests include machine learning algorithms for EDA and VLSI design.

Dr. Xie has received multiple prestigious awards, including the IEEE/ACM MICRO 2021 Best Paper Award, the ACM SIGDA SRF Best Research Poster Award in 2022, the ASP-DAC 2023 Best Paper Award, the ACM Outstanding Dissertation Award in EDA 2023, the EDAA Outstanding Dissertation Award in 2023, and the 2023 Early Career Award from Hong Kong Research Grants Council.

