Transferable Pre-Synthesis PPA Estimation for RTL Designs with Data Augmentation Techniques

Wenji Fang, *Graduate Student Member, IEEE,* Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, *Member, IEEE,* Zhiyao Xie, *Member, IEEE*

***Abstract*—In modern VLSI design flow, evaluating the quality of register-transfer level (RTL) designs involves time-consuming logic synthesis using EDA tools, a process that often slows down early optimization. While recent machine learning solutions offer some advancements, they typically struggle with maintaining high accuracy across any given RTL design. In this work, we propose an innovative transferable pre-synthesis PPA estimation framework named MasterRTL. It first converts the HDL 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 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 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 synthesis results, MasterRTL also excels in accurately predicting layout-stage PPA based on RTL designs and in adapting across different technology nodes and process corners. Furthermore, we investigate two effective data augmentation techniques: a graph generation method and a Large Language Model (LLM)-based approach. Our results validate the effectiveness of the generated RTL designs in mitigating data shortage challenges.**

***Index Terms*—register-transfer level, timing analysis, power modeling, data augmentation**

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 HDL code. In a standard VLSI design flow, designers have to go through the time-consuming subsequent synthesis or even layout stages, relying on full-fledged commercial electronic design automation (EDA) tools to evaluate the design quality based on 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 design quality [2]. While most ML approaches focus on 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., two-dimensional matrices) for analysis with Convolutional Neural Networks (CNNs). However, since the RTL design is in HDL code format rather than conventional data structures, there is no established consensus on the best representation and processing method for them. Some studies [3]–[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]–[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]–[17]. However, these face even greater challenges in generalizing to unseen designs due to the absence of detailed RTL information.

Most recently, several general cross-design ML methods [18]–[22] are proposed to predict design qualities at

Manuscript received XXXX; revised XXXX; accepted XXXX. Date of publication XXXX; date of current version XXXX. This work is partially supported by Hong Kong Research Grants Council (RGC) ECS Grant 26208723, National Natural Science Foundation of China (92364102, 62304192), the Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project, Grant No.2023A03J0013). *(Corresponding author: Zhiyao Xie.)*

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

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

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

the RTL stage. They first convert design RTL to intermediate representations such as the bit-level and-inverter graph (AIG) [18]–[20] or word-level abstract syntax tree (AST) [21], [22], and then develop ML models to evaluate the design quality metrics. Works [18], [19] develop RTL-stage timing models based on neural networks, but these models are limited to combinational circuits and rely on design variations generated by an RTL generator. Work [20] classifies timing paths as critical or non-critical, and then employs the predictions to optimize the logic synthesis results. However, it does not provide concrete timing predictions (e.g., WNS, TNS) and lacks the modeling for power and area metrics. Two recent works [21], [22] utilize the AST representation for RTL code and then evaluate design PPA either based on all register trees [21] or randomly sampled paths [22] extracted from 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 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 pseudo-training paths and the actual paths extracted from the target inference designs.

Furthermore, accurate pre-synthesis PPA modeling necessitates high-quality RTL designs, which are valuable intellectual properties (IPs) for 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]–[25] or by fine-tuning open-source LLMs with Verilog data [26], [27]. The performance of LLMs is typically assessed through benchmarks [28], [29] based on the functional correctness of the generated RTL. However, meeting specific functional requirements is not the primary goal for 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. It is the first work that supports the cross-design RTL evaluation on all major PPA qualities, including both total negative slack (TNS) and worst negative slack (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:

- Q1. What is the most appropriate data format of RTL design (i.e. RTL representation) that best supports ML modeling methods?
- Q2. Based on the RTL representation, how to capture the key patterns to estimate each design objective?
- Q3. How to extend the post-synthesis PPA evaluation across the later layout stage and distinct technology libraries?
- 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]–[20] for 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 design objectives and ML methods.

In response to Q2, since the mechanisms behind ground-truth power, performance, and area 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 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 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 post-synthesis PPA predictions by constructing transfer models that incorporate 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 quality evaluation of the generated RTL code, and then propose a graph-based and a LLM-based data augmentation strategy, which leverage a graph generative model and an LLM, respectively. We demonstrate the utility of generated data in enhancing the model training process, particularly when training data are limited.

Our contributions in this work are summarized below:

- We propose an open-sourced new framework named MasterRTL to efficiently evaluate all PPA values of any

[1]In 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]–[11] are not cross-design, requiring retraining on new designs.

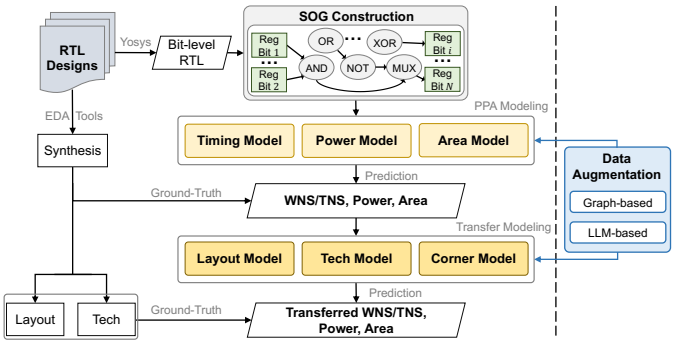
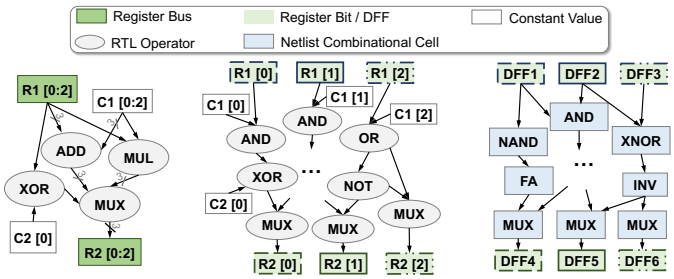


Fig. 1: MasterRTL overall workflow for RTL-stage design PPA prediction with data augmentation techniques.



(a) AST-alike (b) SOG (c) Netlist

Fig. 2: Comparison between different RTL representations and the target gate-level netlist.

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 state-of-the-art solutions.

- Based on the bit-level RTL representation, we customize algorithms for each design objective, capturing their different mechanisms. Among 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.
- We extend the MasterRTL predictions to the layout stage and across various technology libraries with significantly reduced error compared with the original post-synthesis netlist based on the source technology. We further explore the influence of different logic synthesis options.
- 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 HDL-code format of an RTL design $\mathcal{H}$, and the post-synthesis 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 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:

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

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

Rather than the word-level AST, MasterRTL follows [18]–[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: two-input AND, OR, XOR, NOT, and a 2-to-1 MUX, and we name it as simple operator graph (SOG). The process of creating the SOG involves breaking down each multi-bit word into its logic bits and replacing complex word-level operations with their Boolean equivalents, following a pre-set mapping relationship. This SOG generation is achieved through open-source tools like Yosys [30], ensuring a swift and efficient process.

We summarize the key features and advantages of adopting bit-level SOG, in comparison to the AST-alike representation used in [21], [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 pre-synthesis RTL and post-synthesis gate-level netlist.
2) *Generalization*: SOG relies on only 5 fundamental single-bit logic operations, offering broader generalization capabilities than the AST. This simplification reduces variations across different RTL designs and styles

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

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

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 netlists.

Benefiting from the consistent 1-to-1 register mapping of the bit-level representation, we customize the ML models based on distinct PPA mechanisms for 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 paths that have identical starting and ending registers. Our power modeling method propagates toggle rates from registers to 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 prediction accuracy. In the subsequent subsections, 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 multi-stage ML framework to evaluate both the TNS and WNS of any RTL design. The target ground-truth TNS and WNS are obtained from the post-synthesis timing report. Please 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., netlist-stage or layout-stage) [32]–[34], our modeling method will primarily focus on key patterns and use some approximations. Despite the approximations inherently introducing imperfections, we mitigate these by subsequent calibration exploiting ML models.

A primary challenge in RTL-stage timing modeling is that the ground-truth labels are derived from the timing analysis of 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 SOG representation $\mathcal{R}$ and netlist $\mathcal{G}$, and map these paths one by one according to their starting and ending registers. This multi-stage timing modeling process is introduced in detail below.

❶ **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. Not 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 RC (resistance-capacitance) values of standard cells corresponding to the same type, as found in liberty files (e.g., .lib/.db).

❷ **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 $P^{\mathcal{R}}_{i\to j}$ between 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 $P^{\mathcal{R}} i \to j$ 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, $P^{\mathcal{G}} i \to j$, is between register i and j. We collect its ground-truth path delay label using post-synthesis timing analysis EDA tools.

It is important to note that while the critical paths $P^{\mathcal{R}} i \to j$ and $P^{\mathcal{G}} i \to j$ 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 $P^{\mathcal{R}} i \to j$ corresponds to an RTL operator, whereas on $P^{\mathcal{G}} i \to j$, it represents a standard cell.

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

$$f_t^{\text{path}}(P^{\mathcal{R}}_{i\to j}) \;\to\; \text{The path delay of } P^{\mathcal{G}}_{i\to j} \tag{3}$$

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

For the proposed path-level model f_t^{path}, we investigate two ML approaches. Firstly, we employ a transformer model, adopted in large language models (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 accurate path delay prediction, we incorporated the fan-out count of each node in $P^{\mathcal{R}}_{i\to j}$ into our input features. Secondly, we explore a lightweight tree-based model, such as Random Forest [35], with careful feature engineering. The extracted features from $P^{\mathcal{R}}_{i\to j}$ 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 ❶.

❹ **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 netlists. For each register, identified as the endpoint, we capture its critical path. Employing the technique from ❷, we propagate the estimated node delays through the SOG graph. This approach results in the identification of N unique paths, each denoted as $P^{\mathcal{R}}_{*\to j}$ 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 $P^{\mathcal{R}}_{*\to j}$, the

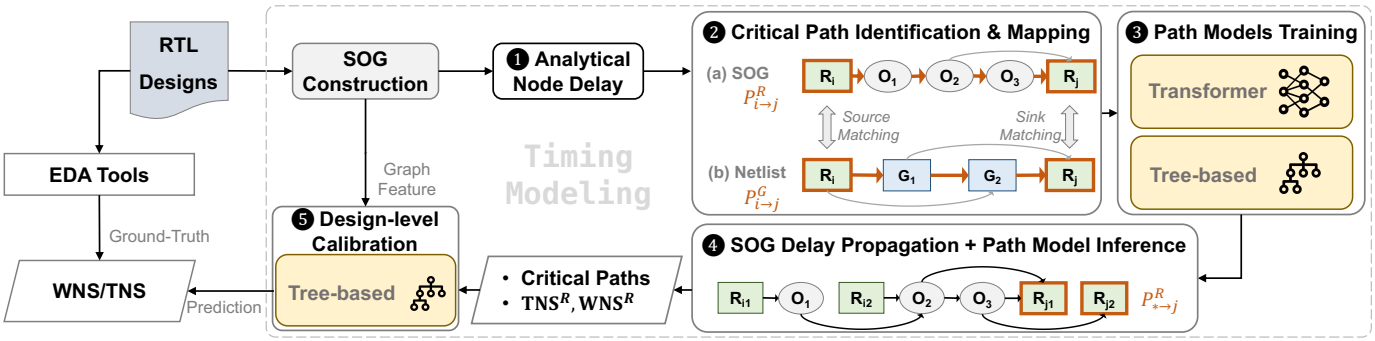


Fig. 3: Timing evaluation flow in MasterRTL. The multi-stage timing model identifies critical paths at the RTL stage, enabling accurate WNS and TNS evaluations.

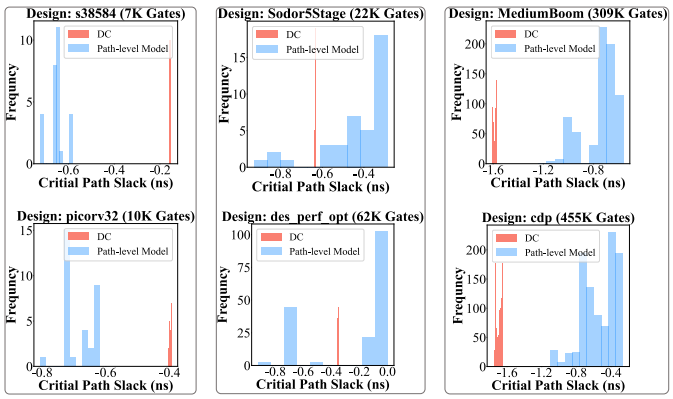
trained path-level model f_t^{path} is employed to predict the delay for each of these paths. Based on these predictions, we calculate the TNS and WNS estimations within the representation $\mathcal{R}$ as follows:

$$\text{TNS}^{\mathcal{R}} = \sum_{j=1}^{N} (clk - f_t^{\text{path}}(P^{\mathcal{R}}_{*\to j}))$$

$$\text{WNS}^{\mathcal{R}} = \min_{j\in[1,N]} (clk - f_t^{\text{path}}(P^{\mathcal{R}}_{*\to j}))$$

❺ **Design-level TNS/WNS calibration.** Given that 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 ❹ are not sufficiently accurate. However, they serve as valuable starting points for further refinement and 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 ($clk - f_t^{\text{path}}(P^{\mathcal{R}}_{*\to j})$) versus the actual slack obtained from the ground-truth netlist timing report. We observe several interesting patterns in Fig. 4:

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



(a) Small Designs (b) Medium Designs (c) Large Designs

Fig. 4: The delay distribution of the worst 1% critical paths from synthesized netlist (DC) and our path-level model.

The observed discrepancy patterns are expected and can primarily be attributed to the optimization processes conducted during logic synthesis. These optimizations are specifically directed towards 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 path-level predictions based on 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 towards 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; 3) the slack distribution of the worst 1% of the N critical paths based on path-level model prediction $clk - f_t^{\text{path}}(P^{\mathcal{R}}_{*\to j})$. For each design, we extract the worst, 10%, 50%, and 90% percentiles of these predicted slacks as features.

C. RTL-Stage Power Modeling

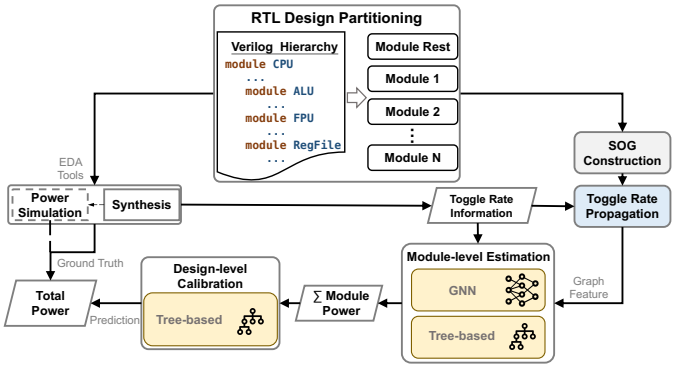


Fig. 5: Power evaluation flow in MasterRTL. The multi-stage power model utilizes the toggle rate information propagated from each RTL module, facilitating precise power evaluation.

For RTL-stage power modeling, we introduce a novel toggle-rate-based and module-level method. Distinct from 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 vector-based and vector-less power simulation scenarios. For vector-based power analysis, 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 toggle rates and, consequently, more precise power analysis. For vector-less power analysis, toggle rates are derived at the beginning of the logic synthesis process using 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 register bits can be sourced from Switching Activity Interchange Format (SAIF) files generated by EDA tools. This method is applicable for both vector-based and vector-less analysis scenarios through 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 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 power data by M times. We estimate the power of each individual module-level partition, denoted as $Power^{\mathcal{G}1}$, $Power^{\mathcal{G}2}$, ..., $Power^{\mathcal{G}M}$. The overall power of the design, $Power^{\mathcal{G}}$, is then calculated by summing the power of all modules: $Power^{\mathcal{G}} = \sum_{i=1}^{M} k_i \cdot Power^{\mathcal{G}i}$, 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: a Graph Neural Network (GNN) model and a lightweight tree-based model. The estimation of total power incorporates both dynamic and static power components, in which the dynamic power is particularly sensitive to toggle rates. For the GNN model, we utilize the sub-SOG converted from 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 6 operator type; 3) propagated toggle rate. In the tree-based approach, feature engineering is performed for each module, primarily based on toggle rate information. The features include: 1) the sum of toggle rate; 2) average toggle rates; 3) the sum of fan-out number multiples toggle rate on each node; 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 power predictions across all 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 operators within the SOG based on the liberty file. These preliminary values, along with 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 post-synthesis PPA labels for a specific technology. However, in real VLSI design scenarios, both the layout stage with physical information and varying technologies significantly impact the final PPA outcomes. Here, we further propose transfer methods to evaluate 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 netlist-stage and layout-stage primarily arise from 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 Sec. III-B.

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

In terms of technology transfer, our approach encompasses not only different technology nodes and but also distinct process corners within each technology. Similar to the layout-stage extension, we evaluate the differences between 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 target and source libraries from 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

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

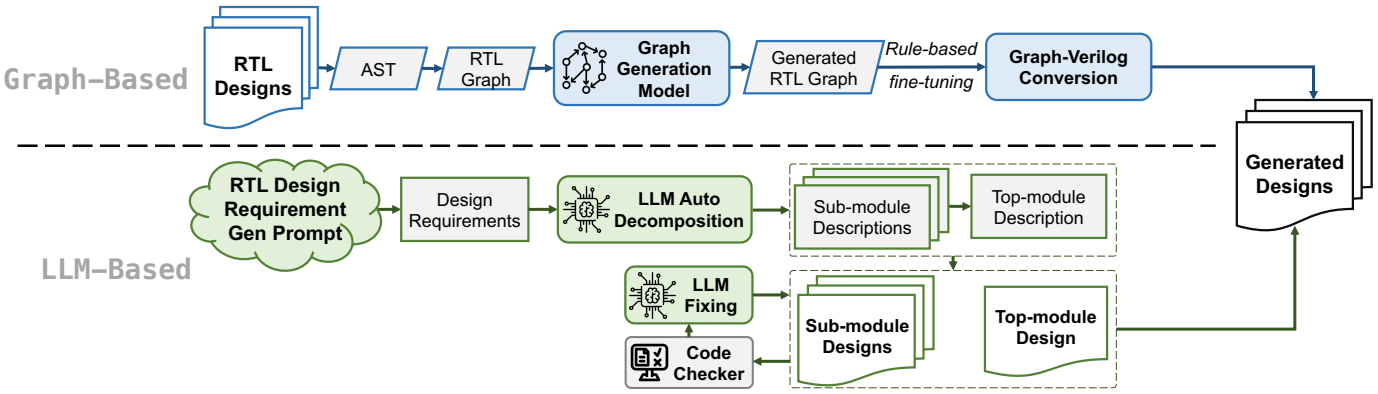


Fig. 6: Two Data augmentation strategies in MasterRTL. The graph-based method utilizes the structural information of RTL designs to generate new RTL graphs. The LLM-based method generates hierarchical RTL codes with various functionalities.

training with 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 sub-modules 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: a graph-based method and an LLM-based approach.

Graph-based data augmentation. Our first strategy leverages the structural characteristics of 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 representation[5] in 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 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 real-world RTL designs using the following predefined rules: (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 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 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 RTL code generation. However, current RTL generation tasks are constrained by specific design descriptions and tend to be suited only for 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 sub-modules, each subsequently implemented in Verilog code. This decomposition is key to creating a

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

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 sub-modules. 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 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 Subsection 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 Results

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 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.

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}

TABLE I: RTL designs for dataset preparation.

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 45nm 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., 22nm, 28nm, 40nm, and 65nm) 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 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], [22]. All experiments are conducted on a server equipped with a 2.9 GHz Intel Xeon(R) Platinum 8375C CPU and 256 GB RAM.

We implemented and evaluated the proposed ML models using the constructed dataset. A 10-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.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y_i}|}{y_i} \times 100\%^{6}, \tag{4}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|, \tag{5}$$

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

The $\bar{y}$ represents the average of the measured values. These metrics bring a comprehensive and fair evaluation of ML models from different aspects, where the higher correlation and lower error indicate better accuracy. Specifically, R measures a linear correlation between predicted values and the 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], [22], which serve as our primary baselines. Additionally, we have implemented a variety of baseline variations to conduct thorough ablation studies. We have endeavored to optimize performance across all 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

[6]To minimize the influence caused by extreme outliers, we cap MAPE values that exceed 100% at 100%, to prevent disproportionate impact on the metric.

Method		ML Model	Description / Hyperparameters
Timing	*Path-level*	Transformer	Adopted from [22], with additional fan-out of the operators as part of the sequence
		Random Forest Regressor	80 estimation tree, maximum depth of 20
	Design-level	XGBoost Regressor	45 estimators, maximum depth of 8
Power	*Module-level*	Graph Neural Network	Graph Convolutional Network, with 2 hidden convolutional layers (10, 70 nodes), 1 sum-pooling layer, Adam optimizer, 0.01 learning rate, 100 training epochs
		XGBoost Regressor	30 estimators, maximum depth of 6
	Design-level	XGBoost Regressor	45 estimators, maximum depth of 8
Area	*Design-level*	XGBoost Regressor	45 estimators, maximum depth of 12
Transfer	*Layout & Tech*	XGBoost Regressor	15 estimators, maximum depth of 8
Generation	*Graph-based*	Graph Neural Network	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

TABLE II: Detailed ML model implementation, including all modeling targets with each internal level or method.

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, light-weight tree-based models are preferred in the intermediate stages of MasterRTL, instead of deep learning-based models.

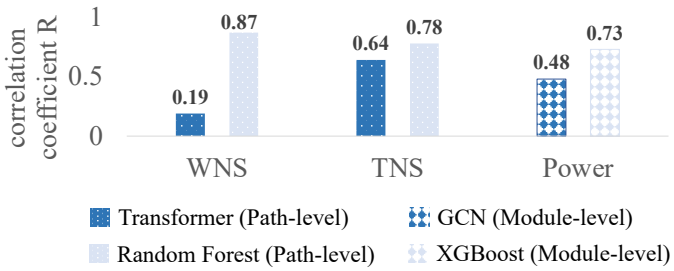


Fig. 7: Evaluation of intermediate-stage ML models in MasterRTL. The lightweight tree-based models outperform the deep learning methods (i.e., Transformer and GCN).

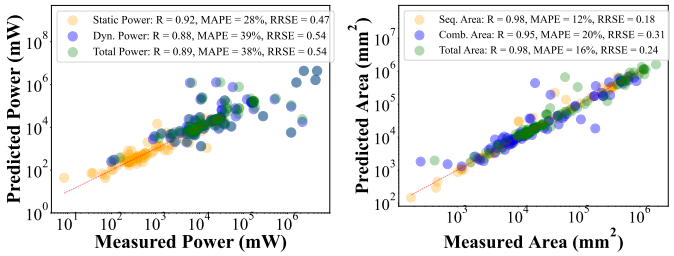


Fig. 8: Accuracy evaluation for individual components of power and area modeling.

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 static power, being independent of 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 sequential and combinational areas. Sequential area, corresponding to registers in the SOG, can be accurately calculated based on the number of 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 compares MasterRTL with previous studies [21], [22], emphasizing the difference in their approaches. As discussed, previous works used AST-like representations, while MasterRTL utilizes SOG. The study by [21] focused on 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 ablation studies.

We have multiple interesting observations in Table III. It reveals that MasterRTL outperforms prior works in all 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 methods using SOG compared to AST-like representations, particularly evident compared with [21]. These findings validate the effectiveness of SOG representation as an ML-friendly RTL representation, suggesting its broader adoption in future studies.

Furthermore, the comparison in Table III indicates that WNS and power estimations are generally more challenging and less accurate compared to 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 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); 3) design-level calibration.

The result of the ablation study in Fig. 10 highlights the importance of specific policies in MasterRTL. For WNS prediction, removing the path-level model, which offers critical path information, results in the most significant accuracy drop. In TNS prediction, the largest accuracy decline is observed when graph features are excluded, underlining the significance of SOG features and design scale information. The accuracy in predicting total power is notably dependent on toggle rate information. When all key policies are removed, MasterRTL's accuracy aligns with the baseline method [21], demonstrating the critical role these policies play in enhancing performance.

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**

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.

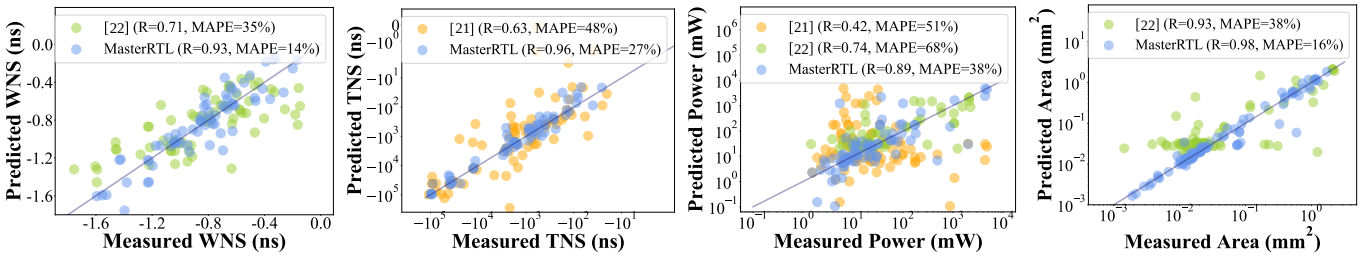


Fig. 9: Comparison between prediction and ground-truth for each PPA metric across all designs. MasterRTL markedly outperforms the SOTA solutions [21], [22] in all PPA evaluations.

Source	Target		Metric	MAE Ori	MAE Ours	MAE IMP	Metric	MAE (K) Ori	MAE (K) Ours	MAE (K) IMP	Metric	MAE Ori	MAE Ours	MAE IMP	Metric	MAE Ori	MAE Ours	MAE IMP
NanGate 45nm	TSMC	22nm	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%
		28nm		4.1	1.7	59%		15.9	3.1	80%		0.544	0.370	32%		0.113	0.011	90%
		40nm		1.7	1.6	6%		5.2	1.5	71%		0.819	0.544	33%		0.030	0.007	78%
		65nm		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%

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.

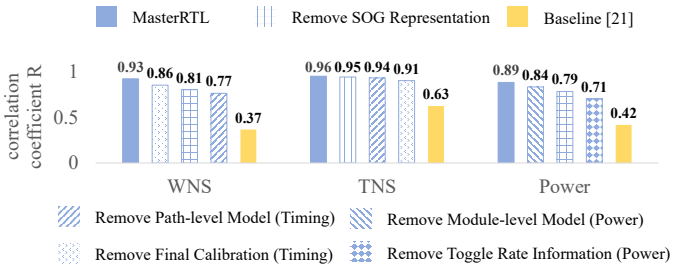


Fig. 10: Ablation study evaluating the contribution of various decomposed factors to the PPA evaluation accuracy.

D. MasterRTL Transfer Modeling Results

Extension to layout stage. In Fig. 11, we first depict the correlation in PPA between the post-synthesis labels (green) from the logic synthesis tool and the post-placement labels (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 post-synthesis PPA predictions (blue) exhibit a similar correlation and MAPE to the post-synthesis labels (green). After extending MasterRTL to the layout stage, the post-placement PPA predictions (red) demonstrate that our framework can achieve satisfactory evaluations for the layout stage early in the RTL stage, even outperforming the post-synthesis labels.

Notably, there is a perfect correlation between post-synthesis and post-placement area labels ($R = 1.0$ and $MAPE = 19\%$). Though 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 post-synthesis design PPA from the source technology, specifically the open-source NanGate 45nm 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:

- For the technology node transfer, our transfer model significantly reduces MAE of all four PPA metrics, highlighting its effectiveness. Notably, the most successful transfer occurs for TSMC 40nm (i.e., lowest MAE), indicating that similarities between technologies lead to more successful outcomes.
- Our transfer model also accurately predicts across dif-

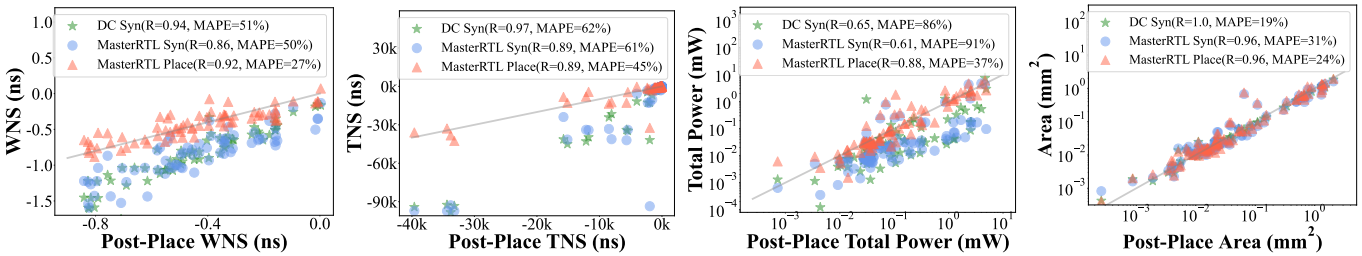


Fig. 11: Extension of MasterRTL for post-placement PPA evaluation. MasterRTL can also provide satisfactory post-placement PPA predictions at the RTL stage.

ferent process corners, maintaining precision in all PPA metrics for corner-to-corner transfers.

- 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 logic synthesis.
- Among different commercial technology nodes, those with similar features, like TSMC 22nm and TSMC 28nm, show a good correlation.
- As for the corner labels, all 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*.

To evaluate the *diversity* of the augmented data, we measure the scale of the design using two metrics: design size and 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.

	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	✗	✔

TABLE V: Quality evaluation of the generated RTL designs.

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 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 generated designs and real designs, we employed the t-SNE plot for visualization, as shown in Fig. 14. Specifically, the BERT language model is employed to convert both real and generated Verilog designs into embedding vectors. These embeddings are then visualized using a t-SNE plot for a thorough comparison. The plot indicates that the embedding distribution of 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 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 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. 13 demonstrates the impact of varying the amount of training data, including both design-level and timing path-level data from real designs, on model accuracy. As indicated by the dashed lines, as the number of 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 training data from real designs across all tasks. The graph-based method is

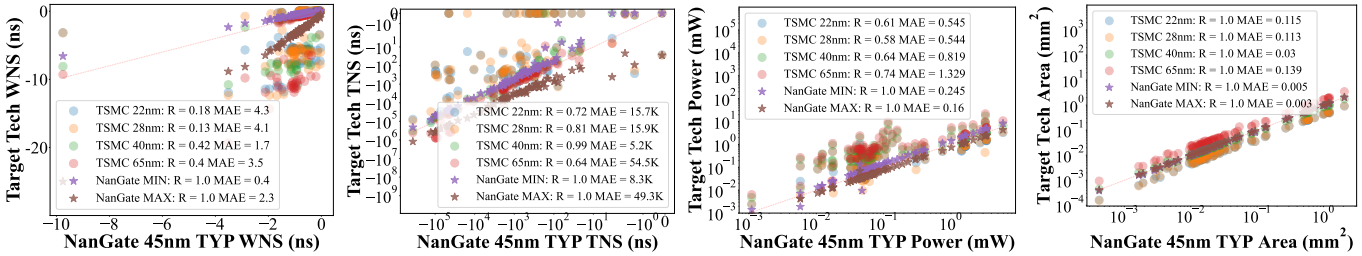
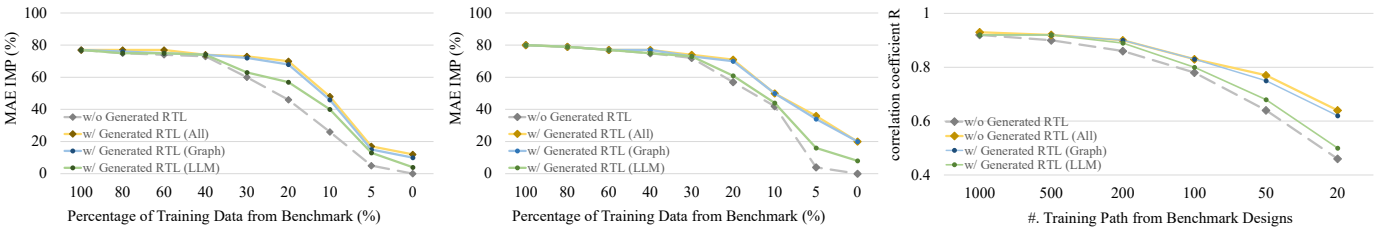


Fig. 12: Visialization of all the target technology nodes and process corners compared with the source library.



(a) TNS transfer (TYP-MIN) (b) TNS transfer (TYP-MAX) (c) Path slack prediction

Fig. 13: 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.

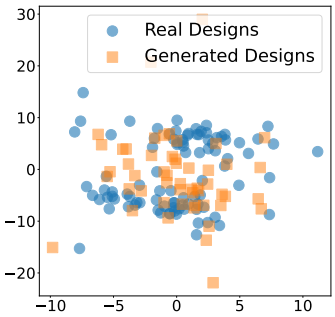


Fig. 14: t-SNE plot visualization for the similarity and diversity between the generated and real RTL designs.

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 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. Discussions

A. Comparison with Open-Source Synthesis Tools

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

% 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 VI: Accuracy of MasterRTL modeling as the proportion of real designs varies, with the total number of training data maintained at a constant 45.

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

TABLE VII: Comparision between open-source logic synthesis tools and MasterRTL.

synthesis tools. Specifically, we employed Yosys and ABC to synthesize the RTL designs and used Prime Time for a fair evaluation of the PPA values of the netlist, with results shown in TABLE VII.

The results indicate that MasterRTL significantly outperforms open-source synthesis tools in terms of both speed and prediction accuracy for all 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 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 paper, we present MasterRTL, a pre-synthesis PPA estimation framework for RTL designs, integrated with innovative data augmentation techniques. Our method incorporates a general RTL representation named simple operator graph (SOG) and customizes multi-stage ML models for WNS, TNS, power, and area separately. MasterRTL not only provides precise post-synthesis 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 data shortage issues.

References

[1] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Design Automation Conference (DAC)*, 2012.

[2] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

[3] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 19–25.

[4] R. Liang, J. Jung, H. Xiang, L. Reddy, A. Lvov, J. Hu, and G.-J. Nam, "Flowtuner: A multi-stage eda flow tuner exploiting parameter knowledge transfer," in *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[5] C. Yu and W. Zhou, "Decision making in synthesis cross technologies using lstms and transfer learning," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 2020, pp. 55–60.

[6] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *DAC*, 2018.

[7] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *54th Annual IEEE/ACM International Symposium on 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 *56th Annual Design Automation Conference (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 *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019, pp. 1050–1062.

[10] Z. Xie, S. Li, M. Ma, C.-C. Chang, J. Pan, Y. Chen, and J. Hu, "DEEP: Developing extremely efficient runtime on-chip power meters," in *Proceedings of the IEEE/ACM International Conference on Computer-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 *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 2015.

[12] W. R. Davis, P. Franzon, L. Francisco, B. Huggins, and R. Jain, "Fast and accurate ppa modeling with transfer learning," in *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 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 *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 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 *Proceedings of the ACM/IEEE Design Automation Conference (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 *Proceedings of the International Conference on Computer-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 Computer Architecture News*, vol. 42, no. 3, pp. 97–108, 2014.

[17] Q. Zhang, S. Li, G. Zhou, J. Pan, C.-C. Chang, Y. Chen, and Z. Xie, "PANDA: Architecture-level power evaluation by unifying analytical and machine learning solutions," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[18] D. S. Lopera, L. Servadei, V. P. Kasi, S. Prebeck, and W. Ecker, "Rtl delay prediction using neural networks," in *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*. IEEE, 2021, pp. 1–7.

[19] D. S. Lopera and W. Ecker, "Applying gnns to timing estimation at rtl," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-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 *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 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 *Proceedings of the 41st IEEE/ACM International Conference on Computer-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 *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, pp. 847–859.

[23] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," 2023.

[24] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[25] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023. [Online]. Available: http://dx.doi.org/10.1109/MLCAD58807.2023.10299874

[26] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, B. Khailany, G. Kokai, K. Kunal, X. Li, C. Lind, H. Liu, S. Oberman, S. Omar, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, W. Turner, K. Xu, and H. Ren, "Chipnemo: Domain-adapted llms for chip design," 2023.

[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," 2023.

[28] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design RTL generation with large language model," 2023.

[29] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[30] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013, p. 97.

[31] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *Proceedings of 2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[32] Z. Xie, R. Liang, X. Xu, J. Hu, C.-C. Chang, J. Pan, and Y. Chen, "Preplacement net length and timing estimation by customized graph neural

network," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2022.
[33] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Design Automation Conference (DAC)*, 2019.
[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 *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1207–1212.
[35] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
[36] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.
[37] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," *Advances in neural information processing systems (NeurIPS)*, vol. 32, 2019.
[38] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1989, pp. 1929–1934.
[39] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *Design & Test of computers (ITC)*, 2000.
[40] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.
[41] VexRiscv, "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," 2022. [Online]. Available: https://github.com/SpinalHDL/VexRiscv
[42] Nvidia, "Nvidia deep learning accelerator," 2018. [Online]. Available: http://nvdla.org/primer.html
[43] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.
[44] *NanGate 45nm Open Cell Library*, https://si2.org/open-cell-library/.
[45] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *Applied Reconfigurable Computing*, ser. Lecture Notes in Computer Science, vol. 9040. Springer International Publishing, Apr 2015, pp. 451–460. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16214-0_42

Wenji Fang received the B.Eng. degree from 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. He is currently a Ph.D. student in the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. 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 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, the 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 a Ph.D. student in the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). 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, Irvine, in 2020. He is currently pursuing a Ph.D. degree with the Department of Computer Science at Duke University. His research interests include AI-enabled hardware design methodologies and architecture for accelerating generative models.

Lisa Wu Wills is an Assistant Professor in the Department of Computer Science and Electrical & Computer Engineering at Duke University. 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. Wills has a Ph.D. in computer science from Columbia University.

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 Electrical and Computer Engineering Department of Princeton University, NJ, USA, in 2021.

He is currently an Assistant Professor with the Microelectronics Thrust, Function Hub of Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, and is also affiliated with the Electronic and Computer Engineering Department of the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong SAR. His research interests include formal verification and hardware model checking.

Zhiyao Xie is an Assistant Professor of the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). Zhiyao received his Ph.D. degree from Duke University in 2022 and B.Eng. from City University of Hong Kong in 2017. His research interests include machine learning algorithms for EDA and VLSI design. He has received multiple prestigious awards, including the IEEE/ACM MICRO 2021 Best Paper Award, ACM SIGDA SRF Best Research Poster Award 2022, ASP-DAC 2023 Best Paper Award, ACM Outstanding Dissertation Award in EDA 2023, EDAA Outstanding Dissertation Award 2023, and the 2023 Early Career Award from Hong Kong Research Grants Council (RGC).