Towards Big Data in AI for EDA Research: Generation of New Pseudo-Circuits at RTL Stage

Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Zhiyao Xie*
Hong Kong University of Science and Technology
Hong Kong, China
{sliudx,wfang838,yludf,qzhangcs}@connect.ust.hk,eezhiyao@ust.hk

ABSTRACT

Machine learning (ML) techniques have demonstrated remarkable effectiveness in electronic design automation (EDA). ML models need to be trained on diverse circuit datasets for better accuracy and generalization capabilities. However, the availability of circuit data remains a long-standing severe issue. The strong data privacy concern in the semiconductor industry makes direct sharing of circuit IPs almost impossible. To address the data availability problem, open-source datasets like CircuitNet have been proposed, but they mostly focus on collecting labels of several existing open-source designs, instead of generating any new designs. In this work, we make an innovative exploration to directly generate new pseudo-circuits without human effort. We believe that generating pseudo-circuits is the most promising, if not the only, approach to achieving "big data" in the semiconductor industry in the foreseeable future. We demonstrate that pseudo-circuits can significantly boost the performance of ML models in early design quality predictions, as early as the pre-synthesis RTL stage.

CCS CONCEPTS

• **Hardware → Software tools for EDA**.

KEYWORDS

Pseudo-circuit, graph generation

ACM Reference Format:
Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Zhiyao Xie. 2025. Towards Big Data in AI for EDA Research: Generation of New Pseudo-Circuits at RTL Stage. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3658617.3697613

1 INTRODUCTION

Artificial intelligence (AI) techniques have demonstrated remarkable effectiveness in electronic design automation (EDA) and agile IC design [11, 21]. For such data-driven technology, access to high-quality, diverse, and representative circuit data is essential for both ML model development and evaluation. However, the lack of circuit data remains a long-standing and primary technical bottleneck. This is largely attributed to the semiconductor industry's strong reluctance to share their circuits, which are valuable commercial IPs. The lack of open *datasets* raises a high barrier to the development of AI for EDA solutions. For ML model training, the label collection process can be highly time-consuming and resource-demanding.

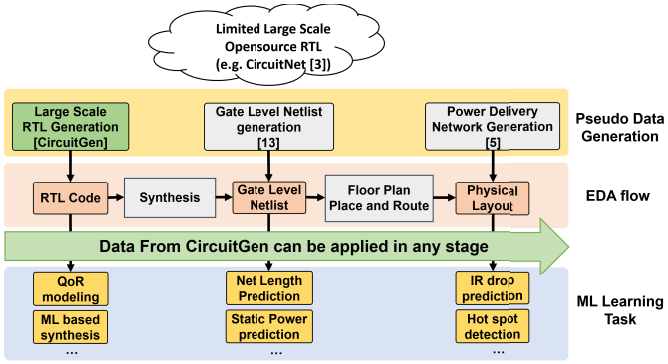


Figure 1: Design augmentation has been developed to generate benchmarks and improve model performance in ML-based EDA tasks. To augment the learning in all the EDA stages, we propose the CircuitGen capable of generating unlimited RTL circuits of varying complexities and scales.

More importantly, limited open-source circuit designs often cannot provide sufficient diversity in training, limiting the ML model performance. As the large language models (LLMs) [6, 15–18] and large foundation models [4] become a trending topic in agile IC design, the bottleneck of circuit availability becomes even more serious.

Several open-source datasets [3, 6, 12, 15, 20] about circuits have been proposed for various circuit design tasks. However, most circuit datasets such as CircuitNet [3, 12] help generate labels of existing open-source circuit designs, instead of generating brand-new circuit designs. In recent years, several works [5, 13] have explored the generation of circuit datasets at the layout stage. However, as Figure 1 shows, these generative methods are limited to circuit layouts, without enforcing new legal circuit functionalities. As a result, for many tasks related to circuit functionality, such as those involving logic synthesis, existing circuit generation methods are not directly applicable.

In this work, we propose a new framework named CircuitGen to generate pseudo-circuits without human effort. We believe that generating pseudo-circuits is the most promising, if not the only, approach to achieving "big data" in the semiconductor industry in the foreseeable future. CirCuitGen proposes a deep learning-based circuit generative model, which not only captures deeper circuit features but also generates new designs in a controllable manner based on user-specified circuit requirements. In addition to the graph connection structure, we also incorporate the type, width, and logic level information of nodes in the register-transfer level (RTL) into the directed graph neural network, providing a more powerful expression ability. Moreover, our proposed logic level-aware graph generation strategy is more effective in generating synthetic graphs that closely resemble the topological structure of real designs. Furthermore, CircuitGen is designed to be capable of generating large-scale circuit graphs with more than 100K nodes. Compared to existing general sequential-based graph generative

*Corresponding Author

ASPDAC '25, January 20–23, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697613

models [10, 14, 27, 28], CircuitGen has significantly lower computational complexity. This is due to our proposed fanout assignment method, which greatly simplifies the process of adding edges.

To demonstrate the benefit of generated pseudo-circuits, we apply them as training data of a highly challenging ML for EDA task involving the processing of circuit functionality: early prediction of area and timing at the RTL stage. This early prediction requires estimating the behavior of logic synthesis tools. Existing ML solutions include [9, 19, 22, 25, 26], and we select the latest MasterRTL [9] and RTLTimer [8] as the standard ML solutions to the problem. By augmenting the training dataset with generated circuits, the ML model accuracy proves to be significantly boosted.

Our data generation method will benefit all practitioners with an interest in AI for EDA techniques, from both EDA and AI communities, both academia and industry. 1) For EDA researchers, open datasets will greatly simplify the development process of AI solutions to their interested problems. Open benchmarks provide a platform for fair and convenient comparison of AI solutions. Researchers are relieved from the tedious circuit collection, data generation, and prior work replication processes. 2) For a large number of AI researchers, this session will attract more of them to contribute new AI algorithms for EDA tasks, reducing the high barrier of circuit design background. 3) For the EDA and design companies, engineers can easily identify and replicate high-quality AI solutions according to the leaderboard of open benchmarks. In addition, the EDA industry can collaborate with academia based on the latest open datasets, if their design IPs cannot be shared.

Contributions in CircuitGen can be summarized below:

- To the best of our knowledge, CircuitGen is the first method that generates new complete pseudo-circuits for ML model training as early as the RTL stage.
- The logic level-aware generation method assisted with ML predictors, user specifications, and predefined constraints is introduced, which not only makes the generated design exhibit a closer resemblance to the real design in terms of its topological structure, but also greatly reduces the time complexity of generating new graphs and makes CircuitGen applicable for large-scale circuit generation.
- Experiments on the graph similarity between generated and original circuits demonstrate that CircuitGen better captures circuit characteristics and generates pseudo-circuits that are closer to real designs.
- Experiments on early RTL-stage area and timing prediction demonstrate that CircuitGen, as a data augmentation method, can help alleviate the data availability problem in AI-based solutions for EDA tasks.

2 PROBLEM FORMULATION

Given an RTL-level circuit graph g_{ref}, generate a series of new circuit graphs $\{g_i\}_{i=1}^{D}$ from scratch. The new graphs must satisfy the internal constraint C_{valid} (a set of rules, e.g., the mux type node must have three inputs, the first one is a selection signal with 1 bit and the other are data signals), allowing them to be converted into RTL code using a parser.

Additionally, we require that these new circuit graphs closely resemble the reference graph in terms of graph local features. Consequently, these newly generated synthetic designs can enhance the performance of machine learning models in EDA downstream tasks when used as training data. In the future, these designs may also serve as benchmarks to test EDA algorithms.

3 CIRCUITGEN OVERVIEW

In this work, we propose CircuitGen, an RTL-level circuit generation framework based on reference graph statistics and user specifications. Figure 2 illustrates the process of CircuitGen in generating pseudo-circuit designs with controllable arbitrary numbers and sizes based on a single existing RTL design. Stage I convert the RTL code into a directed graph representation using predefined rules. Stage II captures the circuit's features using ML models and collects the global statistics (e.g., node type distribution). For each node, Graph Neural Networks are adopted to predict its parents and fanout number. The prediction is only based on the subgraph that contains nodes that precede the given vertex in the topological ordering. Stage III uses the trained ML predictors with the predefined circuit valid constraints, and customized specifications to generate pseudo-circuits from scratch. Finally, the generated circuit graphs are converted back to the RTL code for downstream applications.

We define the "logic level" of a node as the maximum distance from all source nodes (source nodes refer to nodes without parents) that can reach it. We can perform a topological sort on the nodes in the circuit graph. By traversing each node in this sequence, the logical level of a target node is determined by adding 1 to the maximum logical level among its parent nodes.

The graph generation process of CircuitGen starts from the input nodes and proceeds sequentially by logic level. Each time a new logic layer is generated, for each node in that layer, at least one input edge must be connected to a node in the layer above. when generating a new logic layer, we utilize two pre-trained GNN models from the reference design to predict the parents and fanout. In addition, the number of logic layers can be specified, and the distribution of the nodes in the logic layers can also be obtained from the reference design statistics. This generation method ensures that the new circuit is topologically similar to the reference design.

4 METHODOLOGY

4.1 Represent RTL Code with Directed Acyclic Graph in Stage I

The RTL code is in the format of structural language, named hardware description language (HDL), such as Verilog or VHDL. To convert RTL code into a graph representation, we developed a parser based on the open-source tool Yosys to read and compile the RTL code, obtaining a word-level directed graph representation that includes edges, "data" nodes such as "input", "output", "reg", and "operator" nodes such as "adder" and "multiplexer".

Each node has a "type" and "width" attribute after the Yosys compiler. The "type" attribute includes wire, register, adder, etc. For "data" nodes, the "width" attribute represents their bit width, while for "operator" nodes, it represents the bit width of the output and is determined by the inputs. For example, the width of "mux" is the maximum width of its two data inputs, and the width of "cat"(concatenate operator) is the sum.

To facilitate encoding of the circuit, we also need to remove the edges that lead back to registers from a deeper logic level, making the final directed graph acyclic. This is because we will do encoding in a logic-level aware way and the generation process is an autoregressive manner according to the topological ordering. After removing loops from the graph, we add a "logic level" attribute to each node.

Finally, we obtain a directed acyclic graph $G(V, E)$, where V represents the set of nodes $V = \{v_i\}$ with three attributes on each node, E represents the set of edges $E = \{e_{i,j}\}$ and $e_{i,j}$ represents that there exists one edge from vertex i to j. Assume the node number is $|V| = N$ and the edge number is $|E| = M$.

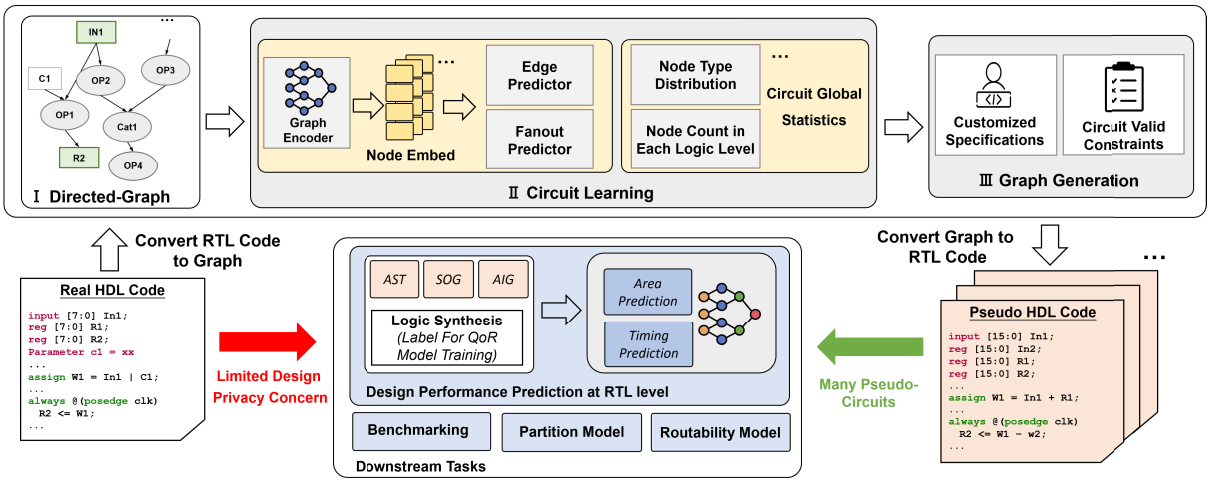


Figure 2: An overview of the CircuitGen workflow.

4.2 Circuit Learning in Stage II

In stage II, we train the predictor of the edge and fanout of newly added nodes by capturing the computation flow of the real circuit.

4.2.1 Motivation of Feature Learning. If simply randomly assign parents for each new node, a significant amount of redundancy will be introduced into the generated pseudo circuit. This redundancy will be removed by logic synthesis tools during optimization. For example, if the single-bit selection signal of a MUX turns out to be a constant value, the other non-select input path will be removed during synthesis. Our subsequent experiments have shown that randomly assigning edges can greatly exacerbate this redundancy. Such redundancy is less common for human-crafted design RTL. Furthermore, randomly assigning edges will fail to capture many local circuit patterns. For example, the majority of circuit nodes only have a single child, while some nodes may have a larger fanout. In CircuitGen, we have employed directed GNN-based *edge* predictors and *fanout* predictors trained on the reference circuit to better capture the circuit features in real design.

4.2.2 Edge Predictor. For a newly added node v_i that has no connections with the graph, the edge predictor can compute the probability of an edge existence between this new vertex and any other node in the graph. Let's assume that the node v_p' embeddings are represented as h_p, and the feature vector of node v_i is represented as z_i. The edge predictor can be formulated as an MLP with a sigmoid activation σ:

$$p(e_{p,i}) = \sigma(MLP_\theta(h_p, z_i)) \tag{1}$$

Where the θ is the MLP parameters and $p(e_{p,i})$ represents the edge existence probability between vertex v_p and v_i. So the current problem is how to compute the feature vector and embeddings for the node. For node features, we adopt the one-hot encoding for the "type" attribute and obtain a T dimension vector t_i. In addition, we perform positional encoding to the "logic level" attribute inspired by the widely used Sinusoidal PE technique [23] in transformers to facilitate learning the positional relationships between different logic layers in the directed graph.

Then the T dimension feature vector z_i for node v_i is obtained by $z_i = t_i + PE_l$, where PE_l is the PE encodings for node in logic level l. To obtain embeddings for each node, we follow the directed graph learning work and update node v as:

$$h_i^{(k+1)} = \sigma\left(h_i^{(k)} + \sum_{j \in \mathcal{P}(i)} \frac{1}{c_i} \cdot W^{(k)} \cdot h_j^{(k)}\right) \tag{2}$$

$h_i^{(k)}$ represents the embedding of node i in the neural network's k layer. The $\mathcal{P}(v)$ represents the parent nodes set of node i.

During the training stage of the edge predictor, for node i in layer l, we extract all nodes with logical levels less than l as $g_{<l}$ and then do graph encoding according to Equation 2. Since the edge existence is a binary value, we sample some nodes from $g_{<l}$ that do not have a connection with node i as negative samples and calculate the loss using cross-entropy.

4.2.3 Fanout Number Predictor. The number of child nodes of a node is referred to as the node's fanout. In digital circuits, fanout is an important characteristic that reflects a node's driving capability and, to some extent, indicates the node's significance within the circuit logic. The fanout predictor uses the node's embedding aggregated from its parents to predict its fanout. Assume we can get the embeddings of the target node as h_i according to Equation 2, then the fanout predictor can be written as follows:

$$Fanout(i) = MLP_\phi(h_i) \tag{3}$$

To predict the number of child node, we can use mean squared error (MSE) loss during the training process.

4.3 Graph Generation in Stage III

In this section, we will introduce our ML predictor-assisted circuit generation process in detail.

4.3.1 Logic-Aware Autoregressive Generation Overview. Our graph generation framework based on logic levels conforms to the inherent logic of circuits, enabling the generation of synthetic graphs that closely resemble the topological structure of the reference circuit design.

Considering the connection relationships between circuit nodes have strong constraints, the generation process should be performed in a step-by-step manner rather than generating the adjacency matrix in one shot [24]. This is because, during the process of stepwise graph expansion, we can identify and correct any invalid connection relationships at any intermediate step. Therefore, we follow the topological order of the circuit nodes to gradually construct the entire graph. The advantage of this approach is that when adding new nodes, we already have the information of the

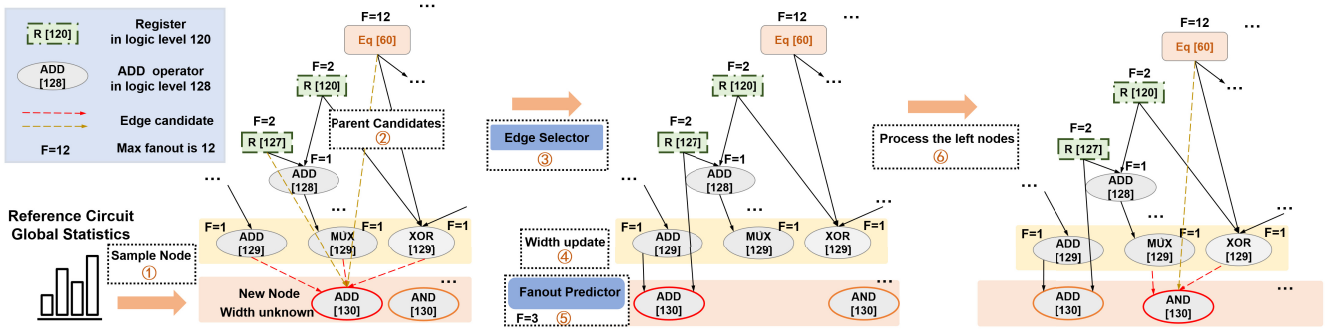


Figure 3: In process ①, we randomly sample a set of nodes with type attributes for the newly added logic layer based on the global information of the reference circuit. In the second step ②, we select parent candidates according to the connection constraints, also ensuring that the current node must have at least one parent node from the previous logic layer. In the third step ③, we use the trained edge predictor to select a parent node from the parent candidates. In steps ④ and ⑤, we update the node fanout and width based on the inputs. The operation for other nodes in this layer follows the same procedure.

entire input cone to predict the node properties, and we only need to assign its parent node from the existing subgraph.

The nodes generated in sequence according to logic levels do not have interconnections, and each node has at least one input edge originating from a node in the preceding logic level. This approach significantly simplifies our generation process. In addition, our proposed logic level-aware autoregressive generation method leverages the topological statistics of the reference design, enabling the generated design to more closely resemble the real design in terms of the graph structure.

4.3.2 Circuit Generation Process in Detail. Given an existing subgraph $g_{<l}$, Figure 3 illustrates the l-th layer generation. In practice, we start the graph construction from the first logic layer containing only "in" nodes.

In process ①, we assign a set of nodes with type attributes to the new logic layer. Based on the global statistics of the reference design obtained in stage II, we extract the distribution of node numbers across each logic level as N_{layer} and the distribution of node numbers for different types as N_{type}. To generate the i-th logic layer (l=130 in Figure 3), we sample from N_{layer} and N_{type}, thereby creating several new vertices with node "type" attributes. It is important to note that at this stage, the newly added nodes lack "width" attributes because their connectivity has not yet been determined.

In the following procedure, We will sequentially process these newly added nodes to determine their parent candidates and width attributes. In process ②, for node ADD, we need to determine a set of parent candidates from the existing subgraph. The criteria for selecting candidates are as follows: Firstly, due to the constraints imposed by the assigned fanout on node outputs, the fanout attribute of the parent node must be greater than its number of child nodes. Secondly, at least one parent must be located in the preceding logic layer which is highlighted as a red dot line. Otherwise, the logic level of the node will no longer be l, conflicting with our assumptions. Lastly, the parent nodes must comply with predefined circuit constraints. For example, different types of nodes have varying fan-in requirements: "reg" nodes require 1 input, "mux" nodes require 3, while "cat" nodes have no restrictions.

In process ③, we use the edge predictor to calculate the edge existing probability based on the embedding of each parent candidate and the feature vector of the newly added node. Based on the obtained probability distribution, we sample a specified number of parent nodes in a stochastic way. It is also important to ensure that the newly added edges comply with predefined rules during this process. For instance, a "mux" node must have at least one single-bit input.

In process ④ and ⑤, we update the bit-width information of the node based on its type and the bit-width of its parent nodes. Additionally, we assign the fanout attribute to the node using the fanout predictor. At this point, the node i (ADD in Figure 3) has been fully processed. Subsequently, we apply the same procedure to the other nodes in this logic layer.

It is important to note that some nodes in the subgraph $g_{<l}$ may reach their fanout limit and thus can no longer be considered as parent candidates as process ⑥ shows. However, the node embeddings in $g_{<l}$ do not need to be recomputed. This is because our graph encoder employs a directed message-passing approach, and there are no connections between nodes within the same logic layers.

4.3.3 The Graph Generation Process Discussion. In the generation process, we can specify the node number and proportion of different types. Moreover, we can also control the logic length and newly generated node number at each logic level. This makes it easier to generate circuits that meet users' specific requirements.

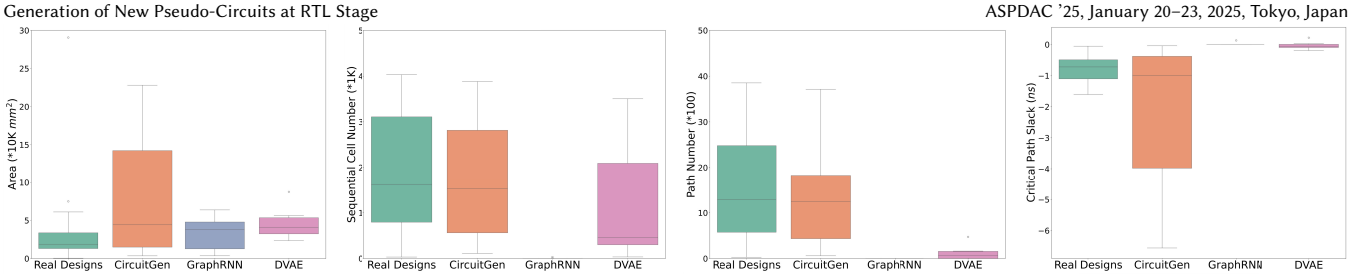
Our proposed generation flow has a better time complexity compared with the representative general graph generation works. In their generation process, each newly generated node requires edge probability prediction with all existing nodes, leading to a time complexity of $O(N^2)$. The generation cost is unaffordable, especially for large-scale graphs such as RV design with around 100K nodes.

By introducing the fanout constraint, we can significantly reduce the number of parent candidates. This is because, in real circuits, the node majority have a fanout of 1. For example, in the graph representation of the TinyRocket design, nodes with a fanout of 1 account for 86.2% of the total, while nodes with a fanout greater than 5 account for only 2.05%.

5 EXPERIMENTAL RESULTS

The register-transfer level (RTL) stage is a crucial early step in modern VLSI design flows, providing maximum optimization flexibility. To meet design specifications, it is essential for designers to optimize their RTL designs adequately at this early stage. Recently, ML-based RTL-level PPA prediction methods have been proposed, which can directly predict the performance of designs without logic synthesis [25] [9] [8]. This can greatly accelerate the design iteration process. However, this application is limited by insufficient open-source RTL code.

The primary objective of this experiment is to utilize the proposed CircuitGen model to generate a set of pseudo-design training



(a)Area (mm^2) (b)Sequential Cell Number (c) Path Number (d) Critical Path Slack (ns)

Figure 4: Netlist statistics for the three synthetic datasets and real benchmarks are presented. The distributions of four types of features: area, sequential cell number, violating path number, and WNS are shown in (a), (b), (c), and (d), respectively. The dataset generated by CircuitGen exhibits *broader feature coverage* and its feature distributions are closer to those of the real designs compared to GraphRNN [27] and DVAE[28].

datasets and to explore the effectiveness of new synthetic circuits for QoR modeling at the RTL level. To the best of our knowledge, this is the first work of applying complete pseudo-RTL designs to machine learning tasks in the EDA frontend stage.

5.1 Experiment Setup

Firstly, we constructed a dataset containing 22 designs with an average size of 13.5K gates based on open-source RTL, which consists of (1) 8 designs in Chipyard [2], (2) 8 designs in Opencores [1], (3) 6 designs in ITC'99 [7]. The dataset encompasses a broad range of digital circuit modules, such as CPU cores and cryptographic units with almost the highest quality available in the open-source community.

To obtain the netlist features after the synthesis stage including design area, reg slack (SL), worst negative slack (WNS), and total negative slack (TNS), Synopsys Design Compiler® 2021 with the NanGate 45nm technology library is used. In order to maintain consistency with real-world scenarios, multiple Design Compiler parameters are used, and the PPA value on the Pareto curve is used as the design ground truth label to ensure optimal trade-offs.

To prepare for the pseudo-data, we not only used CircuitGen to generate a set of synthetic designs but also employed two representative graph generative models, GraphRNN [27] and DVAE [28], to produce two additional datasets for comparison. Since these models were not originally customized for circuit generation, we modified the official code and incorporated circuit constraints to ensure that the generated graphs could be converted into RTL code.

We utilized an Intel(R) Xeon(R) Gold 6438Y+ processor and 8*4090 GPUs as the platform. For CircuitGen, we employed 2-layer and 5-layer message passing for the edge predictor and fanout predictor, respectively. This lightweight model allows for parallel training across the entire graph, enabling model convergence within a few minutes even when using only the CPU. In contrast, GraphRNN [27] is based on RNN and DVAE [28] is based on a sequential variational autoencoder. Their time complexity grows quadratically with the number of nodes, resulting in several days of training time to achieve model convergence, even for relatively small graphs. These graph learning models are all trained on the TinyRocket and aes_cipher_top designs.

To ensure greater diversity in the pseudo-designs, we randomly specify the number of nodes to be between 5K and 20K, and the maximum logic level to be between 200 and 1000. Additionally, we can manually control the ratio of different types of nodes to obtain a wider variety of circuits. Under the same circuit predefined settings, we generated circuits using the three generative models including our CircuitGen.

5.2 Observation for Synthetic Dataset

Figure 4 presents the statistical data of the synthesized circuits from the pseudo datasets generated by the three models, along with the real design benchmark. The statistics include circuit area, sequential cell number, violating path number, and worst negative slack.

From the area statistics, we can observe that although a similar node number is specified for the three-generation models, the graphs produced by CircuitGen still retain a significant area after synthesis. This indicates that, compared to CircuitGen, the graphs generated by the other two baseline methods contained a considerable amount of logic redundancy that was optimized during the synthesis stage.

The design area obtained by CircuitGen is notably larger than that of the real design benchmark. This is because the node number we specified is large, and we did not intend to make the size of the generated circuit similar to the existing benchmark.

In terms of the number of sequential cells, the designs generated by CircuitGen are more similar to the real designs, whereas the designs produced by GraphRNN [27] have most of their registers synthesized away. This is also due to logical redundancy; for instance, registers connected to nodes that are determined to be constant values during logic analysis will be removed.

Additionally, although the graphs generated by DVAE [28] still contain a significant number of sequential cells after synthesis, the very small violating path number indicates that the connection paths of these registers are very short, which does not align with real circuits. On the other hand, CircuitGen still shows no significant gap compared to real designs in terms of the path number metric.

Finally, the WNS metric reflects the longest delay in signal transmission between registers in the netlist, which can partially indicate the length of the paths. We observe that the graphs generated by GraphRNN [27] and DVAE [28] exhibit very small WNS values, failing to capture the delay characteristics inherent in circuits. In contrast, CircuitGen demonstrates a broader coverage of WNS even compared to real designs, indicating that CircuitGen can generate circuit structures with a wide range of WNS values.

A significant gap between RTL design and its netlist can be detrimental to ML-based task learning. This discrepancy arises because these designs may significantly differ from the distribution of real datasets, potentially reducing the model's accuracy. **Considering CircuitGen's ability to achieve broader statistics coverage, it is more suitable to generate and select better pseudo designs in downstream ML-based EDA tasks for data augmentation.**

	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE
No Pseudo-Circuits	WNS	0.86	20 %	0.83	TNS	0.81	50%	0.97	Register Slack	0.7	27%	0.83	Area	0.89	30 %	0.62
GraphRNN [27]		0.88	21 %	0.83		0.80	54 %	0.97		0.7	27%	0.83		0.84	44 %	0.75
DVAE [28]		0.88	24%	0.86		0.78	50 %	0.97		0.69	29%	0.94		0.84	61%	0.96
CircuitGen		0.90	23 %	0.79		1.0	42%	0.60		0.75	17 %	0.72		0.93	22%	0.29

(a) Basic training dataset contains 15 real designs

	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE	Target	R	MAPE	RRSE
No Pseudo-Circuits	WNS	NA	52 %	2.1	TNS	NA	67%	1.1	Register Slack	0.52	34%	1.05	Area	0.65	66 %	1.3
GraphRNN [27]		0.71	42 %	1.7		-0.30	74%	1.1		0.52	34%	1.05		0.51	77 %	1.6
DVAE [28]		0.75	77%	2.6		0.76	93 %	1.1		0.49	36%	1.31		0.70	86%	2.4
CircuitGen		0.88	36 %	1.3		1.0	63%	0.61		0.62	28 %	0.83		0.97	31%	0.46

(b) Basic training dataset contains 5 real designs

Table 1: Model Performance on the WNS, TNS, register slack, and area prediction tasks. The basic training dataset in (a) and (b) contains 15 and 5 real designs respectively. In (a) and (b), the augmentation datasets are added to the basic training set, each always with 25 pseudo-circuits, generated from CircuitGen, GraphRNN [27], and DVAE [28].

5.3 Downstream Tasks at RTL Stage

In this section, we will explore the potential applications of pseudo circuit generation for QoR prediction at RTL-level and we mainly refer to the overall design evaluation (i.e., area, WNS, and TNS) method proposed by MasterRTL [9] and fine-grained timing slack evaluation by RTL-Timer [8].

We use three metrics to measure model performance, They are correlation coefficient (R), Mean Absolute Error Percentage (MAPE), and Root Relative Square Error (RRSE). The lower MAPE and RRSE indicate a better model performance.

We randomly selected 5 and 15 designs in the real benchmark to create two different basic training datasets and randomly selected the 7 designs (Not overlapping with the training set) as the testing set. For each basic training dataset, we augmented it with different synthetic datasets (i.e., three sets of 25 designs each, generated respectively by CircuitGen, GraphRNN [27], and DVAE [28]) to study how these pseudo-designs affect model performance.

5.3.1 Area Prediction. The area prediction accuracy is shown in Table 1. In both basic training dataset settings, models trained on the combined dataset augmented with CircuitGen-generated data always outperformed the model trained solely on real designs and performed the best in all the metrics. It is noteworthy that the models augmented with both DVAE [28]-generated and GraphRNN [27]-generated data performed even worse, regardless of the training dataset setting. This may indicate the significant gap between data generated from the two baselines and the real one due to the logic redundancy.

5.3.2 WNS and TNS Prediction. To implement the overall timing model for WNS and TNS prediction, we follow the two-step procedure described in [9]: (1) training a path-level timing model, and (2) design-level calibration.

The overall timing results are shown in Table 1. Benefiting from the pseudo-designs generated by CircuitGen, the model achieves the best performance across most metrics, attaining an R of 1 for TNS prediction in the training setting with 15 real designs. The results for GraphRNN [27] and DVAE [28] are similar to those for the area, and in many cases, they lead to performance reduction. As observed in Figure 4, the synthetic data generated from GraphRNN [27] and DVAE [28] contain very few paths with large delays. This discrepancy may have caused the model's learning to deviate from the normal timing features.

5.3.3 Register Slack Prediction. In addition to the above overall design quality evaluation, we employ fine-grained register slack prediction for RTL designs [8]. This task is more challenging than overall quality prediction as it requires accurate modeling of the critical timing slack for each register within the logic cone. It can significantly reflect the details (e.g., topological levels and connectivity between register pairs) of our generated RTL designs.

Similar to the analysis of the previous experiments, incorporating synthetic data generated by CircuitGen allowed the machine learning model to achieve the best performance across all metrics. In this experiment, violating paths were extracted from the designs to serve as the training data. However, due to the lack of usable paths from GraphRNN [27], there was no gain to the basic training set. Therefore, the model performance trained on real designs plus GraphRNN [27]-augmented pseudo-circuits is the same as that only trained on real designs. Moreover, although DVAE [28] provided some paths, the significant discrepancy between their delays and the actual distribution still led to poorer model performance in all the cases.

6 CONCLUSION

Data-driven automation in digital circuit design has been widely applied in recent years. However, the number of existing open-source circuits is often limited, and large-scale designs are even rarer. To alleviate this problem, we designed a customized generative model for RTL in a logic-level aware manner. This technique greatly speeds up the graph generation process and makes CirCuitGen applicable for large-scale graph generation. Experimental results demonstrate that CirCuitGen has a better circuit generation ability than existing general graph generative works. Furthermore, the comprehensive analysis of area, WNS, TNS, and register slack prediction tasks further demonstrates that CirCuitGen, as a data augmentation method, helps improve the performance of existing machine learning models.

ACKNOWLEDGEMENT

This work is partially supported by Hong Kong Research Grants Council (RGC) ECS Grant 26208723, National Natural Science Foundation of China (62304192, 92364102), and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Christoph Albrecht. 2005. IWLS 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*.

[2] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro* 40, 4 (2020).

[3] Zhuomin Chai, Yuxiang Zhao, Wei Liu, Yibo Lin, Runsheng Wang, and Ru Huang. 2023. Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).

[4] Lei Chen, Yiqi Chen, Zhufei Chu, Wenji Fang, Tsung-Yi Ho, Yu Huang, Sadaf Khan, Min Li, Xingquan Li, Yun Liang, et al. 2024. The Dawn of AI-Native EDA: Promises and Challenges of Large Circuit Models. *arXiv preprint arXiv:2403.07257* (2024).

[5] Vidya A Chhabria, Kishor Kunal, Masoud Zabihi, and Sachin S Sapatnekar. 2021. BeGAN: Power grid benchmark generation using a process-portable GAN-based methodology. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–8.

[6] Animesh B Chowdhury, Shailja Thakur, Hammond Pearce, Ramesh Karri, and Siddharth Garg. 2023. Towards the Imagenets of ML4EDA. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE.

[7] Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. 2000. RT-level ITC'99 benchmarks and first ATPG results. *Design & Test of computers (ITC)* (2000).

[8] Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. 2024. Annotating Slack Directly on Your Verilog: Fine-Grained RTL Timing Evaluation for Early Optimization. In *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*. ACM, 1–6.

[9] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. 2023. 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, 1–9.

[10] Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. 2020. Graphgen: A scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*. 1253–1263.

[11] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, et al. 2021. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2021).

[12] Xun Jiang, Yuxiang Zhao, Yibo Lin, Runsheng Wang, Ru Huang, et al. 2023. CircuitNet 2.0: An Advanced Dataset for Promoting Machine Learning Innovations in Realistic Chip Design Environment. In *International Conference on Learning Representations (ICLR)*.

[13] Daeyeon Kim, Hyunjeong Kwon, Sung-Yun Lee, Seungwon Kim, Mingyu Woo, and Seokhyeong Kang. 2021. Machine learning framework for early routability prediction with artificial netlist generator. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1809–1814.

[14] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. 2019. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems (NeurIPS)* 32 (2019).

[15] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. 2023. Verilogeval: Evaluating large language models for verilog code generation. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–8.

[16] Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. 2024. RTLCoder: Fully Open-Source and Efficient LLM-Assisted RTL Code Generation Technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[17] Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. 2024. OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation. In *2024 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE/ACM.

[18] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. 2024. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 722–727.

[19] Yikang Ouyang, Sicheng Li, Dongsheng Zuo, et al. 2023. ASAP: Accurate Synthesis Analysis and Prediction with Multi-Task Learning. In *MLCAD*.

[20] Jingyu Pan, Chen-Chia Chang, Zhiyao Xie, and Yiran Chen. 2023. EDALearn: A Comprehensive RTL-to-Signoff EDA Benchmark for Democratized and Reproducible ML for EDA Research. *arXiv preprint arXiv:2312.01674* (2023).

[21] Martin Rapp, Hussam Amrouch, Yibo Lin, Bei Yu, David Z Pan, Marilyn Wolf, and Jörg Henkel. 2021. 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).

[22] Prianka Sengupta, Aakash Tyagi, Yiran Chen, et al. 2022. How Good Is Your Verilog RTL Code? A Quick Answer from Machine Learning. In *ICCAD*.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[24] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. 2022. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734* (2022).

[25] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. 2022. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. 847–859.

[26] Ceyu Xu, Pragya Sharma, Tianshu Wang, and Lisa Wu Wills. 2023. Fast, Robust and Transferable Prediction for Hardware Logic Synthesis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 167–179.

[27] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*. PMLR, 5708–5717.

[28] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. 2019. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in neural information processing systems* 32 (2019).