SynCircuit: Automated Generation of New Synthetic RTL Circuits Can Enable Big Data in Circuits

Shang Liu† *HKUST* sliudx@connect.ust.hk

Jing Wang† *HKUST* jwangjw@connect.ust.hk

Wenji Fang *HKUST* wfang838@connect.ust.hk

Zhiyao Xie* *HKUST* eezhiyao@ust.hk

***Abstract*—In recent years, AI-assisted IC design methods have demonstrated great potential, but the availability of circuit design data is extremely limited, especially in the public domain. The lack of circuit data has become the primary bottleneck in developing AI-assisted IC design methods. In this work, we make the first attempt, SynCircuit, to generate new synthetic circuits with valid functionalities in the HDL format.**

SynCircuit automatically generates synthetic data using a framework with three innovative steps: 1) We propose a customized diffusion-based generative model to resolve the Directed Cyclic Graph (DCG) generation task, which has not been well explored in the AI community. 2) To ensure our circuit is valid, we enforce the circuit constraints by refining the initial graph generation outputs. 3) The Monte Carlo tree search (MCTS) method further optimizes the logic redundancy in the generated graph. Experimental results demonstrate that our proposed SynCircuit can generate more realistic synthetic circuits and enhance ML model performance in downstream circuit design tasks.

I. Introduction

The ever-increasing demands for chip performance have caused escalating integrated circuit (IC) complexity, challenging traditional Electronic Design Automation (EDA) methodologies. In recent years, AI-assisted IC design techniques have demonstrated remarkable potential in accelerating the chip design process. Notable AI for EDA applications include automated chip design generation [1], [2], [3], fast chip quality prediction [4], [5], [6], [7], [8], [9], [10], and automated chip design planning [11].

SynCircuit: Pathway to Big Data in Circuits. Compared with generating datasets with limited circuits in the public domain, we believe the automated generation of a large number of *synthetic* circuits is the most promising way to *completely solve* the circuit data availability problem in the foreseeable future. In this work, we demonstrate the feasibility of this promising direction with a novel synthetic circuit generation framework named SynCircuit.

To the best of our knowledge, SynCircuit proposed in this work is the first technique to automatically generate brand-new synthetic circuits with valid functionalities. Specifically, we achieve this with novel customized graph learning algorithms and follow-up refinement techniques. The synthetic circuits are in the RTL stage and support hardware description language (HDL) code format. They can be automatically synthesized into regular netlists and turned to layouts with commercial tools. These synthetic designs can not only enhance the training and robustness of AI models by providing diverse and extensive datasets but also may serve as benchmarks for assessing the performance of existing design algorithms [12], [9], [13]. In addition, our proposed directed graph generation framework potentially applies to other tasks, such as neural architecture search (NAS) and Bayesian optimization [14], [15], [16].

SynCircuit generates synthetic circuits with 3 phases:

Phase 1. Generation of Large Directed Cyclic Graph (DCG). Circuit design code (i.e., HDL code) can be equivalently represented as directed cyclic graphs (DCG), allowing the utilization of graph generation algorithms to create synthetic circuit designs. However, most machine learning (ML) research on graph generation has focused on undirected graphs [17], [14], [18], [19], [20], leaving unsolved challenges on generating *directed graphs*. For instance, edge predictions in these prior works are determined by applying a *symmetric operator* to the pairwise node embeddings. Also, spectrum-based generation techniques like Spectre [21] and DiGress [22] rely on properties specific to undirected graphs, such as *symmetric Laplacian matrices*. Some works generate directed graphs in an autoregressive manner following the topological order [15], [23]. However, they do not apply to directed *cyclic* graphs, where a topological sort does not exist. In summary, the generation of DCG is not a well-solved problem, not only for synthetic circuit data generation in EDA applications, but also in the general AI community.

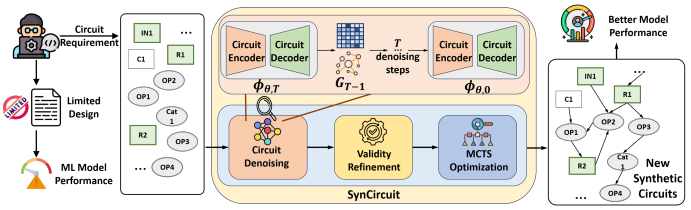


Fig. 1: The overview of SynCircuit. It addresses the serious design scarcity issue by generating new synthetic circuits. Through a three-stage process consisting of denoising, validity refinement, and optimization, SynCircuit can generate an unlimited number of synthetic circuits suitable for downstream ML tasks.

In response to the challenges posed by large directed cyclic graphs in circuit designs, we propose a graph generation algorithm for digital circuits. We employed a diffusion generation framework for graph generation and designed an efficient and low-cost denoising network that includes an encoder and a decoder. For the graph encoder, a directed messaging-passing neural network is utilized to better capture the local features of graphs while only requiring a relatively low memory consumption. For the decoder, we model the directed edge relation based on learnable translated embeddings.

Phase 2 & 3. Refinement of Synthetic Circuits. Additionally, how to ensure the graph validness, i.e., satisfying the circuit constraints, is not also fully explored in graph generation, especially in a one-shot manner. Given the fanin constraints and combinational loop prevention requirements, we propose a probability-guided graph post-processing strategy to further refine the invalid synthetic graph obtained from the denoising process. This technique sequentially processes the fan-in edges of each node, which not only ensures that the resulting circuit satisfies the constraints but also preserves the generative information of the diffusion model as much as possible.

We also point out the logic redundancy problem in the synthetic circuits. To further alleviate the redundancy of the generated designs and improve their utility as data augmentation in downstream tasks, we also employ Monte Carlo Tree Search (MCTS) to further fine-tune the generated graphs. This approach allows us to obtain higher-quality graphs that are better suited for real applications.

We summarize our contributions as follows:

- To the best of our knowledge, we are the first to propose a framework for generating digital circuits at the RTL level from a graph generation perspective. This framework comprises three components in the generation phase: directed cyclic graph generation, probability-guided postprocessing, and MCTS-based optimization.
- We propose a graph generative model solution targeting the challenging directed cyclic graph generation task. We utilize the

† Equal Contribution

∗ Corresponding Author

diffusion framework and reconstruct the edge direction information through an asymmetric edge decoder.
- To further alleviate the redundancy of the generated designs and improve their utility as data augmentation in downstream tasks, we employ MCTS to further fine-tune the generated graphs. This approach obtains higher-quality graphs with optimized logic redundancy which are better suited for real applications.
- We conducted experiments on graph structural property similarity evaluation and demonstrated that SynCircuit generates more realistic graphs. Experiments on early RTL-stage PPA prediction show that SynCircuit can help alleviate the data availability problem in AI-based solutions for EDA tasks.

II. Problem Formulation

DCG representation of HDL code. Given a circuit design HDL code D, it can be mapped to a circuit graph representation G through a bijection function $f : D \leftrightarrow G$ using our developed parser. Here, G is a directed cyclic graph represented as (V, E, X), where V denotes the set of nodes, E represents the set of edges with $e_{i,j} \in E$ indicating a directed edge from v_i to v_j, and X represents the attributes of the nodes. The node attributes include node *type* and *width*. The node types are primarily categorized into IO port, arithmetic operator, register (reg), bit selection, and concatenate operator. The width attribute reflects the output signal width of the corresponding node.

Circuit constraints $\mathcal{C}$. To ensure that the graph can be parsed back into HDL code, valid G must satisfy two types of constraints:

- The node type uniquely determines the number of parent nodes. For example, a node of the type "mux" requires three parent nodes, while the type "add" requires two.
- The graph must not contain any combinational loops. The combinational loop is defined as a cycle that does not include any registers, which would cause timing violations.

Generate new large circuits. In order to provide generation flexibility, we will use the generative model to produce edges E conditioned on the specified node number V and attributes X by users. Our objective is to learn the probability distribution $P(G \mid V, X)$ from a set of real circuit HDL codes through their graph representations $\{G_i\}$. This will enable the generation of a series of new synthetic circuit graphs $\{G_i^{syn}\}$. We aim for these synthetic circuit design graphs to closely resemble real designs in terms of topological structure and satisfy all the circuit constraints $\mathcal{C}$. Moreover, considering most registers in real designs will not be removed in the synthesis stage, we require the generated circuits to have as low logic redundancy as possible.

III. Overview of SynCircuit

Initially, the realistic designs are converted into directed cyclic graph representation. We subsequently trained a diffusion model $P(G \mid V, X)$ on these designs. In the generation stage, as Figure 2 shows, our proposed circuit generation process consists of three phases: initial graph generation, validness post-process, and optimization refinement.

$$P(G) \xrightarrow{❶} G^{ini} \xrightarrow{❷} G^{val} \xrightarrow{❸} G^{opt}$$

In the generation phase of ❶, we adopt the trained diffusion model and apply the denoising process to obtain an initial synthetic graph G^{ini} with a corresponding edge probability matrix $P_E^{(t=0)}$. The value of (i, j) in P_E represents the directed edge $e_{i,j}$ existence probability. In phase ❷, we propose an autoregressive method to determine the edges based on the G^{ini} to ensure that the refined graph G^{val} satisfies all the circuit constraints $\mathcal{C}$. In the final phase ❸, we address the issue of significant logical redundancy in the generated circuits. G^{val} is fine-tuned by the MCTS-based method, reducing logic redundancy and thereby enhancing the performance of downstream tasks. In the following subsections, we will explain each stage in detail.

IV. Phase 1. DCG Generation Technique

Limitation of prior works on DCG generation. Many prior graph generation works utilize a symmetric operator or semi-positive-definite Laplacian matrix to assign edges. This symmetric assumption does not support the asymmetric edge in the directed cyclic graph. For directed acyclic graphs (DAGs), previous work [23], [15] addresses DAG generation by sorting node order based on the topological structure and then generating nodes in a layerwise manner. Topological sorting ensures that a node's parent nodes always precede it in the order. Consequently, the edge direction is automatically determined. However, there is no topological node ordering in the DCG graph since the nodes in a loop share the same topological level.

The proposed DCG generation technique overview. To the best of our knowledge, we are the first to address the task of generating directed cyclic graphs using a diffusion model framework. The method involves a forward diffusion process and a reverse denoising process. In the **forward process**, we gradually corrupt the adjacency matrix A by adding noise, transforming it from the original data distribution into a noisy distribution. The goal of the reverse process is to reconstruct the original adjacency matrix from the noisy version by reversing this corruption. During the **reverse denoising process**, a denoising neural network ϕ_θ with an *encoder* and a *decoder* is trained with real circuits. The *encoder* processes the node attributes and the noisy adjacency matrix to generate node representations that consist of rich graph structural information at each denoising step. The *decoder* utilizes these node representations to effectively reconstruct the ground truth edge connections. By training ϕ_θ in the reverse process, the model learns to progressively denoise the graph structure, ultimately generating synthetic circuits similar to realistic ones.

A. Forward Process

In the forward diffusion process, we progressively corrupt the adjacency matrix A by applying a series of predefined transition matrices, introducing noise at each time step t. Specifically, starting from the initial graph $G^{(0)} = (X, A^{(0)})$, the corrupted adjacency matrix $A^{(t)}$ at time t is obtained via a Markov process:

$$A^{(t)} = A^{(t-1)} \tilde{Q}_A^{(t)}$$

where $\tilde{Q}_A^{(t)}$ is the noise transition matrix at time t. By iteratively applying these transitions, we gradually transform $A^{(0)}$ into a noisy distribution. To control the degree of corruption, we employ a scheduling strategy (e.g., cosine schedule [24]) to adjust the noise level over time. The scheduling ensures that the corrupted adjacency matrices $A^{(t)}$ smoothly transit from the original data distribution to a predetermined noise distribution.

B. Denoising Process

In the denoising diffusion process, our goal is to recover the original adjacency matrix $A^{(0)}$ from the noisy matrix[1] $A^{(T)}$. Based on the node attributes[2] X, we model the conditional probability $p_\theta(G^{(t-1)}) = p_\theta(A^{(t-1)}|A^{(t)}, X, t)$, parameterized by a denoising network $\phi_{\theta,t}$. In the generation stage, the $\phi_{\theta,t}$ will reconstruct a probability matrix P_E^{t-1} given $G^{(t)}$. The new graph G^{t-1} will be updated by sampling the edge existence probability matrix P_E^{t-1}.

C. Encoder Design for Large Graphs

In the denoising network architecture $\phi_{\theta,t}$, the encoder is to capture the structural information of the input noisy graph $G^{(t)}$. Handling large-scale graphs with more than 10K nodes poses computational challenges. To address this, we design an efficient encoder based on Message Passing Neural Networks (MPNNs), which have a computational complexity linear to the number of edges $|E|$.

The encoder generates the representation for each node. When initializing the node embeddings, we use an MLP to obtain a time embedding for the current step t and combine it with the node attributes. By introducing the time step information t which conditions the encoder at different noise levels, we can improve the denoising

[1]In training, the $A^{(T)}$ is obtained from the forward process. In inference, we randomly sample a completely noisy adjacency matrix

[2]In training, the X is obtained directly from the real circuits. In inference, we can either use the $P(X)$ distribution from the real design or set it according to the user's specifications.

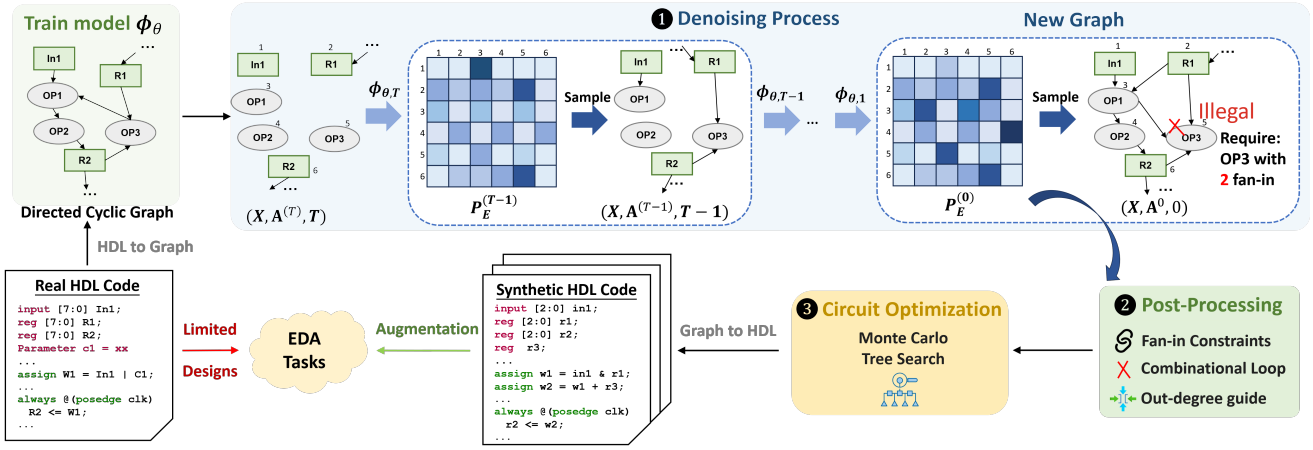


Fig. 2: Overview of SynCircuit framework. The circuit generation process starts with the specified node number V and attributes X. In **Phase 1**, the model performs a reverse diffusion process, where it predicts the probability of an edge existing between any pair (i, j) to obtain $P_E^{(t)}$. Then we sample from $P_E^{(t)}$ to update a graph $G^{(t)}$ as a new time state. After T denoising steps, the $G^{ini} = G^{(0)}$ is obtained. However, the generated graph may not necessarily be valid. In **Phase 2**, during post-processing, we sequentially assign parents to each node based on $P_E^{(t=0)}$, ensuring at each step that the circuit satisfies the constraints. In **Phase 3**, considering that some synthetic circuits exhibit significant logic redundancy, we employ an MCTS-based optimization technique, thereby reducing the generated design redundancy and better serving the downstream tasks.

performance at different diffusion steps. The node representation of each node j is updated through several layers of message passing:

$$H_j^{l+1} = \sigma\left(W_h^l H_j^l + \sum_{i \in \mathcal{P}_j} \frac{1}{|\mathcal{P}(j)|} W_m^l H_i^l\right)$$

where H_j^l is the representation of node j at encoder layer l, $\mathcal{P}(j)$ denotes the parents of node j, W_h^l and W_m^l are learnable weight matrices of the MPNN-based encoder, and σ is Relu function.

D. Decoder Design

In the denoising model $\phi_{\theta,t}$, the decoder will decide the existence of directed edges between pairs of nodes based on their encoded representations H_i from the encoder. In the context of directed graphs, conventional symmetric operations are insufficient to distinguish the directionality of edges. For example, the dot product [25] or Euclidean distance [15] of node embeddings H_i, H_j are commutative (i.e., $H_i^\top H_j = H_j^\top H_i$ and $\|H_i - H_j\|_2 = \|H_j - H_i\|_2$), failing to differentiate an directed edge from node i to j or from node j to i.

To address this limitation, we model the directed edge $e_{i,j}$ by associating H_i and H_j with a learnable relation embedding $r(t)$ [26], capturing the inherent directionality of the edge. In an asymmetric relation, the fundamental idea is that the embedding of the source node H_i, when translated by the relation embedding $r(t)$, is supposed to be close to the embedding of the target node H_j. We define the decoder's prediction for the existence probability of $e_{i,j}$ at time step t as follows:

$$\begin{aligned} P_E^{(t-1)}(i,j) = \hat{p}_{ij}^{(t-1)} &= p_\theta\left(A_{ij}^{(t-1)} = 1 \mid H_i^{(t)}, H_j^{(t)}, t\right) \\ &= \mathrm{MLP}\left(\left\{\left(H_i^{(t)} + r(t)\right) \odot H_j^{(t)}\right\} \oplus d(t)\right) \end{aligned}$$

where $H_i^{(t)}$ and $H_j^{(t)}$ are the representations of nodes i and j at time step t from the encoder, $r(t)$ is the learnable relation embedding obtained by $r(t) = MLP_r(t)$, $\odot$ denotes element-wise multiplication, $\oplus$ denotes vector concatenation, $d(t)$ is the learnable embedding of the time step t obtained by $d(t) = MLP_d(t)$.

V. Phase 2. Probability-Guided Graph Post-Processing

In phase 1, we obtain the G^{ini} and the edge existence probability $P_E^{(t=0)}$. But the G^{ini} will mostly likely violate circuit constraints $\mathcal{C}$. Therefore, we need to utilize the edge existence probability $P_E^{(t=0)}$ to refine G^{ini} and produce designs G^{val} that meet all predefined circuit constraints. We propose a probability-guided sequential post-processing approach to leverage the $P_E^{(t=0)}$ for parent node selection while ensuring both fanin limitations and combinational loop prevention which are required by constraints $\mathcal{C}$.

The post-processing algorithm iteratively processes each node in the circuit graph. For each node i, if its parent edges in G^{ini} are valid, then we will skip this node. If this node violates $\mathcal{C}$, we leverage the predicted edge probabilities from the previous diffusion model, sorting potential parent nodes by their connection probabilities in descending order. Before adding an edge from a candidate parent j, we need to check if the combinational loops would be created by this new edge. It can be achieved by checking if there exists a path from i to j in the subgraph that excludes register-type nodes. This sequential processing continues until the required number of valid parents is found.

VI. Phase 3. Refinement on Circuit Redundancy

Logic redundancy problem. Another important yet rarely studied challenge is logic redundancy. In real circuits, human designers will not likely introduce too much redundant logic into the circuit, which can be reflected by the sequential cell count in the optimized netlist. We define a metric, sequential cell preservation ratio (SCPR) which is calculated by dividing the number of sequential cells in the synthesized netlist by the total number of bits in sequential signals in the pre-synthesis HDL design. Our experiments show that the SCPR is usually between 70% to 100% in real designs. But some synthetic designs G^{val} can not even reach a 10% SCPR, which shows a serious logic redundancy problem.

Our objective in ❸. To better align the generated circuits with real ones not only in the topological aspects but also in the redundancy perspective, we aim to refine G^{val} and obtain G^{opt}, thus reducing the ratio of removed registers by the synthesis tool.

A. Challenges and optimization overview.

To optimize synthesized circuit designs, we face the challenges of an immense exploration space. To tackle this issue, we optimize the driving cone for the target register[3] one by one. In the cone optimization procedure, we introduce an MCTS-based effective strategy for navigating the large design space and finding circuits with better logic preservation quality.

[3]The term "driving cone for a register" refers to the set of nodes obtained by performing a reverse breadth-first search starting from a register node. This search traces back through the parent nodes until nodes of type "const", "in", or other "reg" nodes are encountered. The nodes traversed during this process collectively constitute the driving cone for our target particular register.

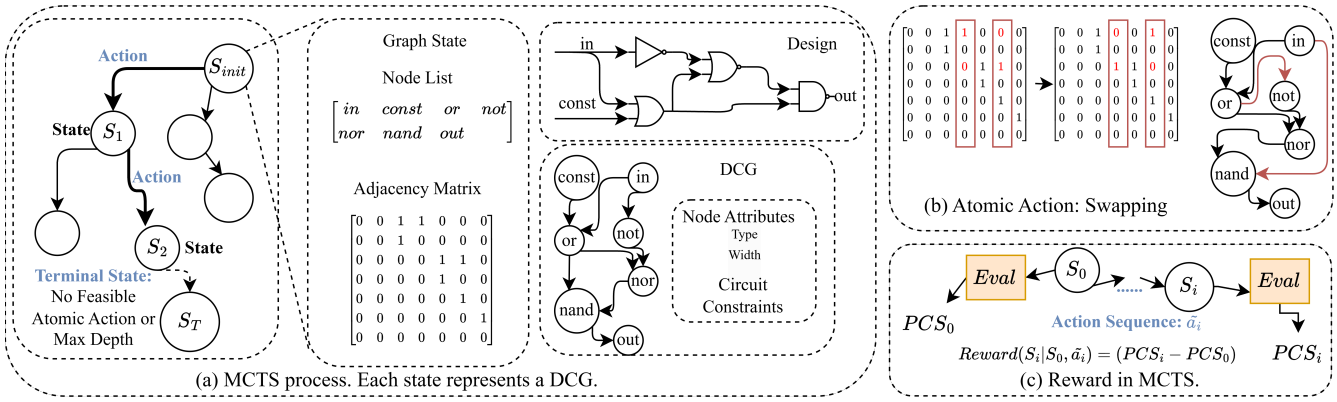


Fig. 3: MCTS-based circuit optimization flow overview. (a) The search tree is on the left and each state node represents an adjacency matrix. (b) A swapping atomic operation on the adjacency matrix facilitates state transformations during the search. (c) We propose a new metric, post-synthesis size (PCS) as the reward model to guide the search process.

B. Monte Carlo tree search-based exploration.

We detail the MCTS process in Figure 3. It guides the exploration and optimization of the design space, driving the search toward solutions with reduced hardware redundancy. Within this framework, each state node represents an adjacency matrix. We define a swapping atomic operation on the adjacency matrix, facilitating state transformations during the search.

Hardware redundancy metric as MCTS reward. To reflect the cone redundancy, we defined the post-synthesis circuit size (PCS). It is calculated by dividing the post-synthesis area by the number of nodes before synthesis. The rationale is that a larger PCS indicates fewer components have been optimized out, implying less redundancy in the synthetic circuit design. The overarching goal of the MCTS is to maximize the PCS while preserving the validity of the circuit.

Action space. The action space for transforming the circuit design during the MCTS process is defined by our proposed atomic swapping operation. For State S_i with $A(i,j) = 1$ and $A(p,q) = 1$, the new state S_{i+1} can be obtained by simply swapping the parents of node j and q, i.e., S_{i+1} with $A(p,j) = 1$ and $A(i,q) = 1$. Each atomic action should be checked if it violates $\mathcal{C}$. The advantage of this atomic operation lies in its ability to ensure that the number of edges in the graph remains unchanged, while also maintaining the out-degree and in-degree of each node in G^{val}.

Selection and expansion. In the selection phase, the MCTS algorithm traverses down the tree, selecting the most promising state which also meets circuit constraints $\mathcal{C}$ to expand based on the UCB1 criterion [27]:

$$a = \arg\max_{a} \left[Q(S,a) + \sqrt{2} \cdot \sqrt{\frac{\ln N(S)}{N(S,a)}} \right]$$

where $Q(S,a)$ is the empirical mean reward of action a in state S, $N(S)$ is the total number of visits to state S, $N(S,a)$ is the number of times action a has been chosen in state S. This formula balances the exploitation of known high-search reward actions with the exploration of less-tried actions. During the expansion step, the MCTS algorithm generates new child nodes by applying the atomic operation to the current node.

Simulation and backpropagation. Due to our objective of identifying the optimal state encountered during the search trajectory rather than optimizing for terminal states, we modify the traditional MCTS simulation and backpropagation mechanisms. In our approach, the reward for each simulation is defined as the maximum state reward $Reward_{max}$ encountered during that search path, rather than the terminal state value. During backpropagation, we update $Q(S,a)$ [27] using the $Reward_{max}$.

VII. Experimental Results

Application of SynCircuit in our experiments. The register-transfer level (RTL) stage offers maximum optimization flexibility in VLSI flow. Recently, machine learning-based RTL-level Power, Performance, and Area (PPA) prediction methods have been proposed, which can directly predict the performance of designs without requiring logic synthesis [6] [5] [28], thereby significantly accelerating the design iteration process. However, the application of these methods is limited by the insufficient availability of open-source RTL code.

Experiments objectives. In this experiment, our primary objective is to explore the effectiveness of new synthetic circuits for PPA modeling at the RTL level. Additionally, we are also interested in investigating the differences between synthetic circuits and real designs from not only the graph structure but also the logic redundancy perspectives. The comprehensive evaluation for SynCircuit may help us gain more insights into the digital circuit generative model.

A. Experimental Setup

Circuit design preparation. Firstly, we developed a dataset comprising 22 designs based on open-source RTL. The detailed dataset information is provided in Table I. It encompasses a diverse range of digital circuit modules, representing some of the high quality designs available within the open-source community.

Source Benchmark	#. of Designs	Original HDL Type	Design Scale (#K Gates) {Min, Median, Max}
ITC'99[29]	6	VHDL	{9, 19, 45}
OpenCores[30]	8	Verilog	{2, 6, 35}
Chipyard[31]	8	Chisel	{12, 19, 52}

TABLE I: Dataset composition and design size information.

Design label preparation. To obtain netlist labels including design area, register slack (SL), worst negative slack (WNS), and total negative slack (TNS), we employed Synopsys Design Compiler® 2021 with the NanGate 45nm technology library. To align with real-world scenarios, multiple parameters within the Design Compiler were adjusted, and a set of the PPA values along the Pareto frontier were utilized as ground truth labels.

Training-testing data splitting. We randomly selected 7 designs from the dataset as the test set, while the remaining 15 designs were used as the training set in the downstream PPA prediction task. To prevent data leakage, all the graph generative models mentioned in this paper were only trained on these 15 training designs.

Graph generative model baselines. We selected several representative and most closely related baselines, including GraphRNN [17], DVAE [15], SparseDigress [32] and GraphMaker [25]. These 4 baselines all need to be adapted to the circuit generation task. GraphRNN [17] and DVAE [15] are node ordering-based autoregressive

Model	1-Wasserstein distance $W_1 \downarrow$						$\mathbb{E}_{\hat{G}\sim p_\theta}\left[\frac{M(\hat{G})}{M(G)}\right] \to 1$					
	Out Degree		Cluster		Orbit		# Triangle		$\hat{h}(A,Y)$		$\hat{h}(A^2,Y)$	
	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core	TinyRocket	Core
GraphRNN [17]	1.03	0.762	0.935	0.136	1.94	0.978	0.016	0.0238	5.33	3.27	2.13	2.53
DVAE [15]	1.31	0.832	0.912	0.146	1.33	1.24	0.241	0.105	6.66	5.19	3.17	2.96
GraphMaker-v [25]	0.678	0.621	0.957	0.112	1.24	1.05	**0.262**	0.501	2.31	1.96	1.53	1.62
Sparse Digress-v [32]	0.598	0.652	0.972	0.135	1.21	0.953	0.163	0.312	4.89	4.32	2.56	2.13
SynCircuit w/o diff	0.373	0.323	0.925	0.0807	1.09	0.926	0.0501	**0.760**	0.629	0.361	0.561	0.321
SynCircuit w/ diff	**0.236**	**0.226**	**0.876**	**0.0452**	**0.344**	**0.231**	0.146	1.34	**0.713**	**0.670**	**0.624**	**0.487**

TABLE II: Evaluation for structural properties similarity with original realistic circuits, best results are in **bold**. Lower W_1 and $\left|\mathbb{E}_{\hat{G}\sim p_\theta}\left[M(\hat{G})/M(G)\right]-1\right|$ reflect a better graph similarity with the realistic ones. All 4 baselines were adapted to generate directed circuit graphs without violating the constraints.

generative approaches. These methods cannot be directly applied to generate directed cyclic graphs. We have to break the cycles in the training circuits and use the topological order of nodes as the sequence for the model training and inference. During generation, edge directions are automatically determined by the topological order. In the sequential generation process, we introduced a validity checker for circuits to ensure that the resulting digital design is valid. **But since these two models can only be applied on DAGs, the generated graph contains no cycles which is very different from the real designs.**

For GraphMaker [25] and SparseDigress [32], these are one-shot approaches that ignore edge direction information, generating only undirected graphs. We applied the Gravity-Inspired Graph Autoencoders [33] method to process the generated undirected graph by assigning the direction for each edge. To ensure the validity of the generated graphs by GraphMaker [25] and SparseDigress [32], we must refine the parent edges in a specific node order. The synthesis tool removed most part of the generated circuits, so these two baseline models can only be used for comparing graph characteristics and cannot support valid designs for downstream tasks.

SynCircuit setup. For our proposed graph generative model, we set the diffusion steps to 9 and utilized a 5-layer MPNN. Both the node attribute embeddings and hidden embeddings were set to 256. In the circuit optimization phase, to accelerate the evaluation process, we replaced the slow synthesis tool with a trained discriminator to approximate the PCS. In the MCTS process, we set the number of simulations to 500 and the maximum exploration depth to 10 for each register cone. All these experiments were conducted on a platform with an Intel(R) Xeon(R) Gold 6438Y+ processor and 8*4090 GPUs.

B. Experimental Results

1) Graph structural properties evaluation: To evaluate the similarity of the generated graph structures with realistic ones, we follow GraphRNN [17] and GraphMaker [25] and report distance metrics for node out degree, clustering coefficient, and four-node orbit count distributions. We use the 1-Wasserstein distance to model statistic distributions from the original and generated graphs, respectively. A lower W_1 value indicates better similarity.

Beyond distribution distance metrics, we also directly compare several scalar-valued statistics [25], [34]. Let $M(G)$ be a non-negative statistic; we report $\mathbb{E}_{\hat{G}\sim p_\theta}\left[M(\hat{G})/M(G)\right]$, where $\hat{G}$ is a generated graph. A value closer to 1 indicates better performance. For the M statistics, we choose triangle count, $\hat{h}(A,X)$ [25], [34] and two-hop correlations $\hat{h}(A^2,X)$. $\hat{h}(A,X)$ reflects the correlation between graph structure and node types.

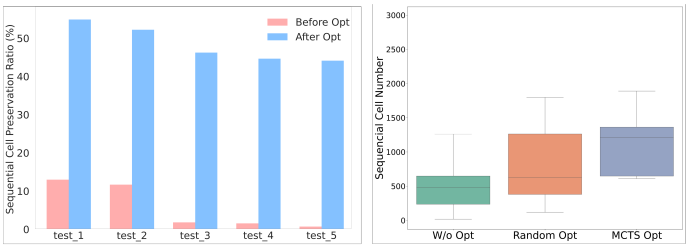
The results are shown in Table II. We can see that our SynCircuit has the best performance among 5 out of the 6 metrics. Benefiting from the diffusion model and the out-degree guidance in the postprocessing phase, the degree distribution of our synthetic circuits is more similar to the real designs. This feature is important because the digital circuits are indeed scale-free networks[4] [35].

[4]A Scale-Free Network is a type of network characterized by a degree distribution that follows a power law. This means that in such networks, the probability $P(k)$ that a node has k connections (or degree k) decreases polynomially as k increases

SynCircuit w/o diff is an ablation study where we remove the diffusion model and randomly construct edges when generating G^{ini} and $P_E^{(0)}$, but applying post-processing to ensure certain predefined constraints. We observe that its performance is greatly inferior to the complete SynCircuit, demonstrating the effectiveness of our designed generative diffusion model.

2) Logic redundancy and timing evaluation: We employ Monte Carlo Tree Search (MCTS) to optimize the register cones within G^{val}. For comparison, we implement an ablation study of randomly altering edge connections on G^{val} while still ensuring every step is valid. We utilize the same number of simulations as MCTS and adopt the optimal solution identified throughout the process as the results. The SCPR enhancement is illustrated in Figure 4(a), demonstrating that MCTS can significantly reduce logic redundancy.

Figure 4(b) displays the number of sequential cells saved during the logic synthesis using different optimization techniques. We can see that by introducing our MCTS-based refinement, the registers saved in the netlist have been greatly improved over no optimization and the optimization performance is also better than that using a random optimization method.



(a) SCPR before and after MCTS-based optimization

(b) No. of Sequential Cell Saved in the Netlist Stage

Fig. 4: (a) Logic redundancy metric SCPR is calculated by dividing the number of sequential cells in the synthesized netlist by the total number of bits in sequential signals in the pre-synthesis HDL design. Here we select the five G^{val} examples with the most significant logic redundancy. The register ratio (SCPR) cannot even reach 20% if we do not optimize G^{val}. After the MCTS optimization, the SCPR is greatly improved and exceeds 50% in some synthetic circuits. (b) The number distribution of registers preserved after logic synthesis. The logic redundancy is greatly reduced by our proposed MCTS optimization method.

Netlist timing statistics comparison We use two metrics to evaluate the design's timing characteristics: Worst Negative Slack (WNS) and Total Negative Slack divided by the number of violated paths (TNS/NVP).

- WNS represents the most severe timing violation in the design, indicating the largest negative slack among all timing paths, which points to the longest delay and the most critical bottleneck in signal propagation.

Model	Register Slack			WNS			TNS			Area		
	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓
Basic training data (No-pseudo circuits)	0.70	27%	0.83	0.86	20%	0.83	0.81	50%	0.97	0.89	30%	0.62
GraphRNN [17]	0.70	27%	0.83	0.88	21%	0.83	0.80	54%	0.97	0.84	44%	0.75
DVAE [15]	0.69	29%	0.94	0.88	24%	0.86	0.78	50%	0.97	0.84	61%	0.86
SynCircuit w/o opt	0.72	24%	0.79	**0.90**	23%	0.85	0.80	48%	0.86	0.86	39%	0.72
SynCircuit w/ opt	**0.77**	**16%**	**0.70**	0.89	**20%**	**0.80**	**0.97**	**45%**	**0.64**	**0.95**	**25%**	**0.34**

(a) Basic training dataset contains 15 real designs. Lower $|R-1|$, MAPE and RRSE reflect better model prediction performance.

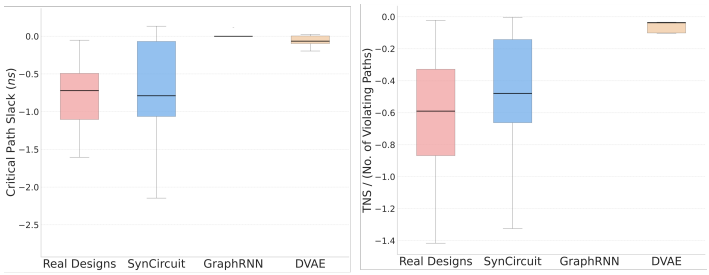
Model	Register Slack			WNS			TNS			Area		
	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓	R → 1	MAPE ↓	RRSE ↓
Basic training data (No-pseudo circuits)	0.52	34%	1.1	NA	52%	2.1	NA	67%	1.1	0.65	66%	1.3
GraphRNN[17]	0.52	34%	1.1	0.71	42%	1.7	-0.30	74%	1.1	0.51	77%	1.6
DVAE[15]	0.49	36%	1.3	0.75	77%	2.6	0.76	93%	1.1	0.70	86%	2.4
SynCircuit w/o opt	0.56	35%	1.2	0.65	47%	1.9	0.72	70%	0.85	0.63	62%	1.1
SynCircuit w/ opt	**0.67**	**25%**	**0.76**	**0.87**	**34%**	**1.2**	**0.98**	**61%**	**0.63**	**0.95**	**36%**	**0.61**

(b) Basic training dataset contains 5 real designs. Lower $|R-1|$, MAPE and RRSE reflect better model prediction performance.

TABLE III: Model Performance on the Register Slack, WNS, TNS, and area prediction. 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 SynCircuit, GraphRNN [17], and DVAE [15]. R with NA means that the model prediction is the same for all testing designs

- The metric TNS/NVP provides the average negative slack per violated path, reflecting the overall severity of timing violations and offering insights into the distribution of delays.

From Figure 5, We observe that the graphs generated by GraphRNN [17] and DVAE [15] exhibit very small WNS (critical path slack) and TNS/NVP values, failing to capture the inherent delay characteristics of circuits. In contrast, our SynCircuit demonstrates a more similar distribution of the two metrics to real designs. This suggests that SynCircuit is more effective in modeling the diverse timing behaviors present in real circuits.



(a) Critical Path Slack (ns) (b) TNS / No. of Violating Paths (ns)

Fig. 5: Netlist statistics for the three synthetic datasets and real benchmarks. The distributions of two statistic metrics: Critical Path Slack (WNS) and the ratio of Total Negative Slack to the Violating Path Numberare shown in (a) and (b), respectively. The dataset generated by SynCircuit exhibits *more similar statistics distributions* to those of the real designs compared to GraphRNN [17] and DVAE[15].

3) SynCircuit application in RTL tasks: In this part, we explore the application of synthetic circuit generation for ML-based RTL-level PPA prediction model, primarily referencing the overall design evaluation methods (i.e., area, WNS, and TNS) proposed by MasterRTL [5] and the fine-grained timing slack evaluation by RTL-Timer [28]. We employ three metrics to assess model performance: correlation coefficient (R), Mean Absolute Percentage Error (MAPE), and Root Relative Squared Error (RRSE). Lower MAPE and RRSE values indicate better model performance.

We randomly selected 5 and the full 15 training real designs to create two basic training datasets. For each basic training dataset, we augmented it with different synthetic datasets (three sets of 25 designs each), generated respectively by SynCircuit, GraphRNN [17], and DVAE [15], to study how these synthetic designs affect model performance.

The results are shown in Table III. We can see that in both training settings, models trained on datasets augmented with Synthetic-generated data consistently outperformed those trained solely on real designs, achieving the best results across all metrics. And we have more model performance gain when the basic training dataset contains 5 designs. The Register Slack MAPE is reduced by 10% in both basic training settings. And we have a 30% area MAPE reduction in the 5 basic training dataset settings.

Notably, models augmented with DVAE [15]-generated and GraphRNN [17]-generated data performed worse, regardless of the training dataset size. This may indicate a significant gap between the data generated by these baselines and the real designs due to logic redundancy. As observed in Figure 5, the synthetic data generated from GraphRNN [17] and DVAE [15] contain very few paths with large delays. This discrepancy may have caused the model's learning to deviate from the normal timing features for Register Slack, WNS, and TNS tasks.

We also included non-optimized circuits SynCircuit w/o opt (G^{val}) as part of an ablation study. It can be observed that the excessive design logic redundancy introduced may have adversely affected the model's predictive performance, rendering it inferior to scenarios involving only real training circuits. This observation indirectly demonstrates the importance of optimizing logic redundancy in generating synthetic circuits that more closely resemble real-world designs.

VIII. Conclusion And Future Work

Data-driven automation techniques have become increasingly prevalent in digital circuit design in recent years. Nevertheless, the availability of open-source circuits is often limited. To address this issue, we propose the SynCircuit, an automatic digital circuits generation framework. It consists of three stages: directed cyclic graph generation, probability-guided postprocessing, and MCTS-based optimization. Our experiments demonstrate that the designs generated by SynCircuit not only exhibit structural properties that closely resemble those of real designs, but also enhance model performance in ML-based power, performance, and area (PPA) prediction tasks at the RTL early stage.

IX. Acknowledgement

This work is supported by the Hong Kong Research Grants Council (RGC) ECS Grant 26208723, CRF Grant C6003-24Y, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and the National Supercomputing Center in Guangzhou Nansha Subcenter for computational resources.

REFERENCES

[1] Z. Pei, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[2] C.-C. Chang, Y. Shan, S. Fan, J. Li, S. Zhang, N. Cao, Y. Chen, and X. Zhang, "Lamagic: Language-model-based topology generation for analog integrated circuits," *arXiv preprint arXiv:2407.18269*, 2024.

[3] 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," *arXiv preprint arXiv:2312.08617*, 2023.

[4] Y. Bai, A. Sohrabizadeh, Z. Qin, Z. Hu, Y. Sun, and J. Cong, "Towards a comprehensive benchmark for high-level synthesis targeted to fpgas," *Advances in Neural Information Processing Systems*, vol. 36, pp. 45 288–45 299, 2023.

[5] 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.

[6] 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.

[7] Z. Qin, Y. Bai, A. Sohrabizadeh, Z. Ding, Z. Hu, Y. Sun, and J. Cong, "Cross-modality program representation learning for electronic design automation with high-level synthesis," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024, pp. 1–12.

[8] S. Yang, Z. Yang, D. Li, Y. Zhang, Z. Zhang, G. Song, and J. Hao, "Versatile multi-stage graph neural network for circuit representation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 313–20 324, 2022.

[9] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "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.

[10] J. Zou, X. Wang, J. Guo, W. Liu, Q. Zhang, and C. Huang, "Circuit as set of points," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[11] C. Bai, J. Huang, X. Wei, Y. Ma, S. Li, H. Zheng, B. Yu, and Y. Xie, "Archexplorer: Microarchitecture exploration via bottleneck analysis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 268–282.

[12] M. Yoon, Y. Wu, J. Palowitch, B. Perozzi, and R. Salakhutdinov, "Graph generative model for benchmarking graph neural networks," *arXiv preprint arXiv:2207.04396*, 2022.

[13] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, "A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda," *arXiv preprint arXiv:2504.03711*, 2025.

[14] S. Jiang, D. Jiang, M. Balandat, B. Karrer, J. Gardner, and R. Garnett, "Efficient nonmyopic bayesian optimization via one-shot multi-step trees," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 039–18 049, 2020.

[15] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs," *Advances in neural information processing systems*, vol. 32, 2019.

[16] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "Brp-nas: Prediction-based nas using gcns," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 480–10 490, 2020.

[17] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *International conference on machine learning*. PMLR, 2018, pp. 5708–5717.

[18] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer, 2018, pp. 412–422.

[19] 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*, vol. 32, 2019.

[20] X. Chen, J. He, X. Han, and L.-P. Liu, "Efficient and degree-guided graph generation via discrete diffusion modeling," *arXiv preprint arXiv:2305.04111*, 2023.

[21] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer, "Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators," in *International Conference on Machine Learning*. PMLR, 2022, pp. 15 159–15 179.

[22] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard, "Digress: Discrete denoising diffusion for graph generation," *arXiv preprint arXiv:2209.14734*, 2022.

[23] M. Li, V. Shitole, E. Chien, C. Man, Z. Wang, Y. Zhang, T. Krishna, P. Li *et al.*, "Layerdag: A layerwise autoregressive diffusion model of directed acyclic graphs for system," in *Machine Learning for Computer Architecture and Systems*, 2024.

[24] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International conference on machine learning*. PMLR, 2021, pp. 8162–8171.

[25] M. Li, E. Kreačić, V. K. Potluru, and P. Li, "Graphmaker: Can diffusion models generate large attributed graphs?" *arXiv preprint arXiv:2310.13833*, 2023.

[26] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Advances in neural information processing systems*, vol. 26, 2013.

[27] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[28] W. Fang, S. Liu, H. Zhang, and Z. Xie, "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, 2024, pp. 1–6.

[29] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers (ITC)*, 2000.

[30] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[31] 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, 2020.

[32] Y. Qin, C. Vignac, and P. Frossard, "Sparse training of discrete diffusion models for graph generation," *arXiv preprint arXiv:2311.02142*, 2023.

[33] G. Salha, S. Limnios, R. Hennequin, V.-A. Tran, and M. Vazirgiannis, "Gravity-inspired graph autoencoders for directed link prediction," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 589–598.

[34] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 887–20 902, 2021.

[35] X. F. Wang and G. Chen, "Complex networks: small-world, scale-free and beyond," *IEEE circuits and systems magazine*, vol. 3, no. 1, pp. 6–20, 2003.