# LayerDAG: A Layerwise Autoregressive Diffusion Model of Directed Acyclic Graphs for System

Mufei Li*, Viraj Shitole*, Eli Chien*, Changhai Man*
Zhaodong Wang†, Srinivas Sridharan‡ Ying Zhang†, Tushar Krishna*, Pan Li*
*Georgia Institute of Technology. *mufei.li@gatech.edu, viraj2@gatech.edu, ichien6@gatech.edu, cman8@gatech.edu, tushar@ece.gatech.edu, panli@gatech.edu*
†Meta. *zhaodongwang@meta.com, zhangying@meta.com*
‡NVIDIA. *srisridharan@nvidia.com*

***Abstract*—Directed acyclic graphs (DAGs) are ubiquitous in the design and optimizations of systems. For example, neural networks have become a key computational workload for system design, and neural architectures are natively DAGs. Intermediate representations in compilers or hardware-synthesis tool to characterize execution dependencies and dataflows of computation also often take the form of DAGs. For sensitive scenarios, we believe that *learning* a conditional generative model of DAGs allows releasing synthetic data that preserves downstream utility while protecting intellectual property (obfuscation). In addition, such models can efficiently search the space of valid DAGs for desired properties, which is of great potential use to applications like compiler optimization. However, generating realistic DAGs is challenging due to their inherent directional and logical dependencies. This paper introduces LayerDAG, an autoregressive diffusion model designed to address these challenges in DAG generation. By iteratively removing the nodes without predecessors and their outgoing edges, we can obtain a unique tokenization that turns a DAG into a sequence of directed bipartite graphs and its nodes into a sequence of node layers. LayerDAG leverages autoregressive generation to model directional dependencies and employs diffusion models to capture logical dependencies within each bipartite graph. Empirical studies demonstrate that LayerDAG outperforms existing DAG generative models, particularly for generating large-scale DAGs with up to 400 nodes—a critical scenario for system. Our implementation will be available at https://github.com/Graph-COM/LayerDAG.**

## I. Introduction

Directed acyclic graphs (DAGs) have broad applications in ML system design. With directed edges, they can naturally depict dataflows between operators in neural network model architectures, enabling neural architecture search [8]. Further, by treating subtasks as nodes and task dependencies as edges, DAGs provides a compact representation for optimizing the scheduling of task execution [31], compiler optimizations for hardware accelerators like TPUs [1], [27], and for optimizing circuit netlists [9].

There is a growing interest in leveraging AI models for AI/ML system design/optimization [12], [16], [29], and was in fact a key thrust of the Architecture2.0 Workshop. This necessitates learning the structure of DAG datasets corresponding to the underlying design-space (e.g., hardware design, compiler optimization, and so on) and generating representative DAGs for the optimized solution. To this end, this work presents a generative model for synthesizing DAGs. We believe a generative model for DAGs serves several purposes. First, by learning the distribution of DAGs, a generative model allows sampling realistic synthetic DAGs that reflect the real data distribution. Second, in sensitive scenarios like system and hardware design, this enables preserving data utility while protecting intellectual property (**obfuscation**) [15], [32], [44]. Third, a conditional DAG generative model would be capable of efficiently searching the space of valid DAGs for property optimization [43]. For instance, consider distributed training of a foundation model with data parallelism [7]; replicas of a model can perform similar operations on a huge number of GPUs, and the model itself often consists of repetitive layers. This would make an execution trace of the program [21], [32] extremely large due to redundant and repetitive patterns. In such cases, a generative model may learn to identify common components and compress traces by generating smaller synthetic ones [5].

Serving as an abstraction for flows and node dependencies, DAGs pose significant challenges in developing powerful and efficient generative models due to their intrinsic strong directional and logical dependencies, such as control flows, logic gates, and dimension requirements of matrix operations. These complexities are further magnified in large-scale DAGs, presenting a unique combination of challenges regarding both scale and logical rules.

Motivated by the aforementioned challenges, we present LayerDAG, a layerwise autoregressive diffusion model of DAGs. To tackle the directional logical dependencies, inspired by the strong capability of language models in modeling sequential logic, we propose autoregressive generation with the following tokenization. By iteratively removing the nodes without predecessors and their outgoing edges, we observe that a DAG induces a unique ordered partition of the nodes, which we refer to as layers (Fig. 1), along with a partition of the edges into bipartite graph structures. LayerDAG performs autoregressive generation at the granularity of layers while following the direction of edges. To model the node and edge dependencies, we adopt diffusion models that have achieved remarkable generation quality in various domains [28], [36].

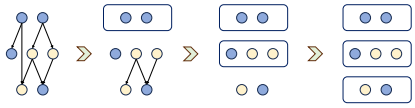

Fig. 1. Each DAG has a unique layerwise partition.

Fig. 2. The two edges are equally likely, but mutually exclusive. Naïvely predicting their existence by independently sampling from two Bernoulli distributions can violate the mutual exclusivity.

Specifically, the diffusion model is conditioned on previously generated bipartite graphs to generate the next bipartite graph consisting of node attributes and edges. Essentially, our model is the first autoregressive diffusion model for DAG generation (Fig. 3).

Our model advances existing DAG generative models in multiple aspects. Methodologically, although autoregressive models have been adopted by D-VAE [43] and GraphP-NAS [18] for DAG generation, they treat either a single node or a node set of constant size as a token. This tokenization method imposes an order between nodes that should be incomparable in the partial order defined by the directed edges, violating the inductive bias inherent in the DAG structure. We argue that the violation may hurt the generalization capability of generative models. DiffusionNAG [2] employs diffusion models to generate only node attributes given a DAG structure. Diffusion models have been used to generate undirected graphs [14], [25], [36], but they ignore the directional information in DAGs, while our work demonstrates the necessity of the autoregressive component in modeling directional dependencies in DAGs. From the application perspective, all the existing works focus on generating small DAGs (with #nodes $\leq$ 24) for neural architecture search (NAS), while our model is capable of generating much larger flow graphs (up to $\sim$ 400 nodes) for system and hardware.

We conduct extensive experiments to verify the effectiveness of our model. To examine the model capability in learning strong directional logical rules, we construct a challenging synthetic dataset. To evaluate the conditional generation capability for the purpose of data sharing with obfuscation, we consider three real-world datasets – computational graphs on TPU, flow graphs on FPGA, and neural architectures deployed on edge devices. Each dataset contains thousands of DAGs, with individual DAGs comprising up to hundreds of nodes. On all four datasets, LayerDAG consistently achieves a better generation quality than the best baselines, by an average margin of 9.1%.

## II. Background

Discrete Denoising Diffusion Probabilistic Model (D3PM) [3] is a diffusion model for discrete data. It has two phases, a forward diffusion process and a reverse process. Let $\mathbf{Z}^{(0)} \in \mathbb{R}^{M\times C}$ be the one-hot encodings of a categorical attribute with $C$ possible values for $M$ real samples. The forward process corrupts them into purely random samples by progressively changing the category of more samples over $T$ consecutive time steps. To corrupt $\mathbf{Z}^{(t)}$ into $\mathbf{Z}^{(t+1)}$, it computes and samples from a conditional distribution $q(\mathbf{Z}^{(t+1)}|\mathbf{Z}^{(t)}, t) = \mathbf{Z}^{(t)}\mathbf{Q}^{(t+1)}$, where $\mathbf{Q}^{(t+1)} \in \mathbb{R}^{C\times C}$ is a pre-determined transition matrix. A denoising network $\phi_\theta$ is trained to predict the uncorrupted data $\hat{\mathbf{Z}}^{(0)}$ from $(\mathbf{Z}^{(t)}, t)$. Composing the transition matrices across multiple time steps yields the closed-form expression $q(\mathbf{Z}^{(t+1)}|\mathbf{Z}^{(0)}, t) = \mathbf{Z}^{(0)}\overline{\mathbf{Q}}^{(t+1)}$, where $\overline{\mathbf{Q}}^{(t+1)} = \mathbf{Q}^{(1)}\mathbf{Q}^{(2)}\cdots\mathbf{Q}^{(t+1)}$. During the reverse process, the trained denoising network can be used to convert $\mathbf{Z}^{(T)}$ drawn from the prior distribution into realistic data.

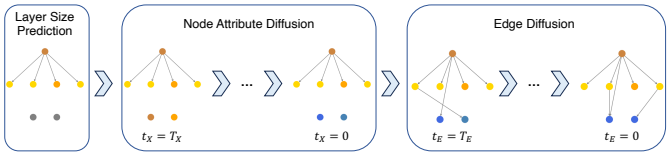


Fig. 3. Layerwise autoregressive diffusion.

DiGress [36] extends D3PM for non-autoregressive generation of undirected graphs with a single categorical node attribute. It treats the presence or absence of an edge between a node pair as a binary attribute. GraphMaker [19] extends DiGress for generating multiple categorical attributes.

## III. LayerDAG

In this section, we present the LayerDAG framework. We first describe the unique layerwise DAG partition, which naturally leads to a layerwise tokenization for autoregressive DAG generation. To perform layerwise set-level predictions of node attributes and edges, we propose layerwise diffusion that is capable of modeling complex logical rules via refinement with multiple denoising steps. By adopting permutation equivariant and invariant model components, LayerDAG is also permutation invariant, which benefits model generalizability and robustness. As the cost of DAG generation is proportional to the constant number of denoising steps, we propose to alleviate the cost with a number of denoising steps proportional to the layer depth for a flexible quality-efficiency trade-off.

### A. Layerwise Partition and Tokenization

Consider an arbitrary DAG denoted by $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \cdots, N\}$ is the node set, and $\mathcal{E} = \{(u_e, v_e)\}_e$ is the set of directed edges. Assume the DAG has a set of nodes without predecessors, i.e., the nodes are not destinations of any edges, and we denote them by $\mathcal{V}^{(1)}$. Iteratively, we take $\mathcal{V}^{(l+1)} \subset \mathcal{V} \setminus \mathcal{V}^{(\leq l)}$ to be the set of nodes whose predecessors are in $\mathcal{V}^{(\leq l)}$, where $\mathcal{V}^{(\leq l)} = \bigcup_{i=1}^{l} \mathcal{V}^{(i)}$. It follows that $(\mathcal{V}^{(1)}, \mathcal{V}^{(2)}, \cdots, \mathcal{V}^{(L)})$ forms an ordered partition of $\mathcal{V}$ the moment $\mathcal{V}^{(L+1)} = \emptyset$ (Fig. 1), which we refer to as layers. For each layer depth $1 \leq l \leq L-1$, we also take

$\mathcal{E}^{(l+1)} = \{(u_e, v_e) \in \mathcal{E} | u_e \in \mathcal{V}^{(\leq l)}, v_e \in \mathcal{V}^{(l+1)}\}$, and $(\mathcal{E}^{(2)}, \mathcal{E}^{(3)}, \cdots, \mathcal{E}^{(L)})$ forms an ordered partition of $\mathcal{E}$. Note that $(\mathcal{V}^{(\leq l)} \cup \mathcal{V}^{(l+1)}, \mathcal{E}^{(l+1)})$ is essentially a bipartite graph. This layerwise partition naturally extends to arbitrary node and edge attributes. The partition exists if $|\mathcal{V}^{(1)}| > 0$, which is guaranteed for a DAG. Furthermore, this way of construction is unique and allows reconstructing the original DAG from the sequence $(\mathcal{V}^{(1)}, \mathcal{V}^{(2)}, \mathcal{E}^{(2)}, \cdots, \mathcal{V}^{(L)}, \mathcal{E}^{(L)})$. The process of converting a raw data sample into a sequence is known as tokenization in the context of generative models, with notable examples being subwords for language models [30] and patches for image generation [26]. Essentially, layerwise partition leads to a layerwise tokenization method for DAGs.

### B. A Layerwise Autoregressive Generation Framework

Motivated by the layerwise tokenization, we propose a layerwise autoregressive framework that emulates the partition process for the purpose of DAG generation. Let $G^{(\leq l)}$ be a partially generated DAG with $l$ layers, and $G^{(\leq 0)} = G^{(0)}$ be the initial empty graph. To generate the $(l+1)^{\text{th}}$ layer, it first predicts the number of new nodes with $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$. Then it generates the node attributes with $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$. Finally, it generates the edges with $p_\theta\left(\mathcal{E}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$. The generation process terminates when a layer size of 0 is predicted.

### C. Layerwise Diffusion

The modeling of $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$ and $p_\theta\left(\mathcal{E}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$ requires simultaneously sampling all set elements. Naïve sampling of set elements from multiple conditionally independent distributions cannot model the intra-set constraints like mutual exclusivity (Fig. 2). This is particularly an issue for generating real-world DAGs with heavy hard logical rules like systems and circuits, and the rule violations can accumulate over layers of generation.

To tackle this issue, we adopt discrete denoising diffusion for multiple rounds of set refinement. We use the empirical marginal distribution of a categorical attribute $\mathbf{m} \in \mathbb{R}^C$ as its corresponding prior distribution. The composed transition matrix is chosen to be $\overline{\mathbf{Q}}^{(t)} = \overline{\alpha}^{(t)}\mathbf{I} + \left(1 - \overline{\alpha}^{(t)}\right)\mathbf{1}\mathbf{m}^\top$, where $\overline{\alpha}^{(t)} = \cos^2\left(\frac{\pi}{2}\frac{t/T+s}{1+s}\right)$ is the cosine noise schedule [24], $\mathbf{I} \in \mathbb{R}^{C\times C}$ is the identity matrix, and $\mathbf{1} \in \mathbb{R}^C$ is the one-valued vector. We employ two separate diffusion processes for node attribute and edge generation (Fig. 3), as suggested by Li et al. [19]. For simplicity, we focus on categorical node attributes in this paper. Our approach can be extended to handle real-valued attributes [11], [37].

Autoregressive layerwise diffusion may be more efficient than non-autoregressive counterparts for DAG generation. Non-autoregressive diffusion models use a constant number of denoising steps in generation. A single denoising step involves refining the whole graph structure and node attributes, which is already very costly. In contrast, autoregressive layerwise diffusion only refines node attributes and edges for one layer in a single denoising step, and the total number of denoising steps naturally scales with respect to the number of DAG layers.

### D. Implementation

$p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$, $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$, and $p_\theta\left(\mathcal{E}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$ are shared across layers, and involve a DAG encoder. We use an off-the-shelf bidirectional message passing neural network (MPNN) [38], which allows efficient parallel training over DAGs and layer depths. A single bidirectional MPNN layer updates node representations with synchronous message passing over both the directed edges and their revered counterparts: $\sigma\left(\mathbf{A}\mathbf{H}\mathbf{W}_{\text{forward}} + \mathbf{A}^\top\mathbf{H}\mathbf{W}_{\text{reverse}} + \mathbf{H}\mathbf{W}_{\text{self}}\right)$ , where $\sigma$ is a non-linear layer, $\mathbf{A}$ is the adjacency matrix for a DAG, $\mathbf{H}$ is the node representation matrix, and $\mathbf{W}$'s are learnable weight matrices. Both layer size prediction $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$ and node attribute prediction $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$ involve computing graph representations with a set pooling operator like sum or mean over the updated node representations.

To generate $\mathbf{X}^{(l+1)}$, the node attribute prediction module first samples $\mathbf{X}^{(l+1,T_X)} \in \mathbb{R}^{|\mathcal{V}^{(l+1)}|\times C}$ from its prior distribution, where $T_X$ is the maximum number of denoising steps. Then iteratively, it samples $\mathbf{X}^{(l+1,t)}$ with a denoising network $\phi_{\theta_X}\left(G^{(\leq l)}, \mathbf{X}^{(l+1,t+1)}, t+1\right)$ for $t = T_X - 1, T_X - 2, \cdots, 0$. To predict the set of denoised node attributes, $\phi_{\theta_X}$ integrates the representations of $G^{(\leq l)}$ and $t+1$ into the embeddings of $\mathbf{X}^{(l+1,t+1)}$, and then employs a set transformer, a transformer without positional encodings [35], over them for the final predictions. Similarly, the edge prediction module iteratively samples $\mathcal{E}^{(l+1,t)}$ with a denoising network $\phi_{\theta_E}\left(G^{(\leq l)}, \mathbf{X}^{(l+1)}, \mathcal{E}^{(l+1,t+1)}, t+1\right)$, which augments $G^{(\leq l)}$ by $(\mathbf{X}^{(l+1)}, \mathcal{E}^{(l+1,t+1)})$ for computing the node representations of $\mathcal{V}^{(\leq l+1)}$. To predict the probability of edge $(u, v) \in \mathcal{V}^{(\leq l)} \times \mathcal{V}^{(l+1)}$, it concatenates and transforms the representations of node $u$, node $v$, and $t+1$ with an MLP.

Bidirectional MPNNs and transformers without positional encoding are permutation equivariant, while set poolings are permutation invariant. Consequently, all three modules are permutation invariant, and hence LayerDAG is also permutation invariant. In theory, sufficiently powerful permutation non-invariant models are capable of memorizing the training data without capturing the underlying directional and logical dependencies. Consequently, they may have much worse performance in generating unseen DAGs than LayerDAG (**generalizability**). This is particularly an issue for applications like obfuscation, where the goal is to release useful synthetic DAGs in replacement of the original DAGs.

### E. Layer-Index-Based Denoising Schedule

While refinement with multiple denoising steps improves the generation quality, the amount of time taken for generation also grows linearly in the number of denoising steps. Training multiple LayerDAG models with a variety of denoising steps does natively enable a trade-off between generation quality and efficiency, but is cumbersome and costly. A natural follow-up question is: Can we achieve flexible quality-efficiency trade-off with a single trained LayerDAG model?

As $l$ increases, both $|\mathcal{V}^{(\leq l)}|$ and $|\mathcal{E}^{(l+1)}|$ increase in general, resulting in more complex edge dependencies. To effectively

handle this pattern, we introduce a non-uniform denoising schedule for better allocation of the time budget. Specifically, we propose to set the total number of denoising steps to linearly increase in $l$.

$$T^{(l)} = T_{\min} + \lfloor (T_{\max} - T_{\min}) \cdot \min\{l/L_{\max}, 1\} \rfloor \quad (1)$$

where $T_{\min}$ and $T_{\max}$ are the minimum and maximum number of denoising steps to use during generation and allow users to make a flexible quality-efficiency trade-off. $l$ is the current layer index, $L_{\max}$ is the maximum number of layers in the training data, and $\lfloor \cdot \rfloor$ is the floor function.

## IV. Related Work

GraphRNN [41] and Li et al. [20] propose autoregressive models to sequentially generate an undirected graph, one node/edge at a time. D-VAE [43] extends this approach to DAGs, and adopts topological orderings of nodes based on empirical studies. Although topological orderings respect the constraint of edge directions, they are not unique. Consequently, unlike LayerDAG, autoregressive models using topological orderings are not permutation invariant. For better efficiency, GRAN [22] proposes to sequentially generate node sets of a constant size and their incident edges, where the edges for the new node set are predicted simultaneously using a mixture of Bernoullis, potentially with intra-set connections.

## V. Experiments

With empirical studies, we aim to answer the following questions. **Q1**) In terms of generation quality, how well does LayerDAG perform against the existing methods when applied to DAGs with hard logical rules and real-world applications in system and hardware design? **Q2**) Does layer-index-based denoising schedule enable an effective trade-off between quality and efficiency? If so, how well does LayerDAG perform against the existing methods with a comparable time budget?

### A. Datasets and baselines

**Datasets.** We construct a synthetic dataset of latent preferential DAGs (LP) with hard logical constraints including mutual exclusivity for layer size, node attributes, and edges, which allows directly evaluating the proportion of the generated DAGs that are valid (**validity**). Each node is associated with a binary label 0 or 1. The first layer of a DAG consists of a single node, whose label is drawn from a Bernoulli distribution with $p = 0.5$. Let $n_i^{(l)}$ be the number of nodes with attribute $i$ in layer $l$ for $i \in \{0, 1\}$. $n_i^{(l+1)}$ is sampled from $\mathcal{U}\left(\max\left\{0, n_i^{(l)} - (l+1)\right\}, n_i^{(l)} + (l+1)\right)$. Each new node can have one or two predecessors, and at least one of them belongs to the previous layer. In the case of two predecessors, they need to have distinct labels (**mutual exclusivity**).

In addition, we repurpose and adapt three representative real-world datasets, originally developed for DAG property prediction, to serve as testbeds for conditional DAG generation. The datasets are associated with computation workloads executed on diverse hardware platforms, and they well fit the end scenario of synthetic data sharing for system/hardware benchmarking. Originally released as part of the TpuGraphs dataset, **TPU Tile** is a collection of kernel graphs for machine learning workload on Tensor Processing Units (TPUs), with graph labels indicating the runtime averaged over a set of compilation configurations [27]. **High-level synthesis (HLS)** is a collection of data flow intermediate representation graphs for compiled C programs, with each DAG labeled according to the resource usage of look up table measured on Field Programmable Gate Arrays (FPGAs) [39]. **NA-Edge** is a collection of DAGs representing neural architectures, with labels indicating their inference latency on mobile CPU [8], [42]. Table I presents the dataset statistics.

TABLE I
Dataset statistics. $|\mathcal{V}|$, $|\mathcal{E}|$, and $L$ are averaged over graphs. $|\mathcal{V}^{(l)}|$ is averaged over layers.

| Dataset | # graphs | $\lvert\mathcal{V}\rvert$ | $\max\lvert\mathcal{V}\rvert$ | $\lvert\mathcal{E}\rvert$ | $\max\lvert\mathcal{E}\rvert$ | $L$ | $\max L$ | $\lvert\mathcal{V}^{(l)}\rvert$ | $\max\lvert\mathcal{V}^{(l)}\rvert$ | # attributes |
|---|---|---|---|---|---|---|---|---|---|---|
| TPU Tile | 6, 301 | 40.8 | 394 | 42.9 | 711 | 11.2 | 72 | 3.6 | 21 | 1 |
| HLS | 2, 062 | 88.6 | 356 | 110.7 | 477 | 27.75 | 78 | 3.2 | 28 | 7 |
| NA-Edge | 2, 000 | 231.1 | 339 | 265.8 | 385 | 149.1 | 185 | 1.5 | 4 | 14 |

Performing ground truth evaluations for conditional generation of DAGs in system and hardware design requires direct measurements on specific computational platforms. For example, the HLS dataset requires program implementation and measurement on FPGAs [39]. Such evaluations are computationally costly or infeasible due to limited access. Additionally, they demand specialized domain knowledge that often exceeds the expertise of general machine learning practitioners. Recently, employing ML-based surrogate cost models has emerged as a popular and effective alternative to direct measurement in various system and hardware optimizations [4], [6], [10], [13], [17], [23], [27], [33], [34], [45]. In light of these achievements, we propose to evaluate the quality of generated DAGs with ML-based surrogate models. Specifically, we partition the real labeled DAG datasets into training/validation/test subsets. After developing a DAG generative model using the training and validation subsets, we use the real training and validation labels as conditions for DAG generation. The generated labeled DAGs essentially form synthetic training and validation subsets. Inspired by previous practices [19], [40], we develop two DAG property prediction models based on bidirectional MPNN using the same automated pipeline respectively on the real and synthetic train/val subsets. We then compare the performance of the two models on the real test set using Pearson correlation coefficient. A generative model is considered better if its corresponding model achieves a performance closer to that of the model developed on the real subsets.

**Baselines.** GraphRNN, D-VAE, and GRAN were previously introduced in Section IV. Both GraphRNN and GRAN originally tackle the generation of non-attributed undirected graphs, and we extend them for attributed DAGs following the existing practices [18], [43], including the use of random topological orderings due to superior empirical performance. To compare the capability in set generation of LayerDAG against GRAN,

TABLE II
EVALUATION RESULTS. BEST RESULTS ARE IN **BOLD**. THE PEARSON VALUE OBTAINED WITH THE REAL DATA IS ALSO LISTED FOR REFERENCE.

| Model | LP | TPU Tile | HLS | NAS |
|---|---|---|---|---|
| | Validity ↑ | Pearson (Real: 0.761) | Pearson (Real: 0.976) | Pearson (Real: 0.996) |
| LayerDAG | **0.907** | **0.676** | **0.831** | **0.994** |
| D-VAE | 0.640 | 0.437 | 0.796 | 0.898 |
| GraphRNN | 0.367 | 0.619 | 0.751 | 0.989 |
| GRAN | 0.117 | 0.307 | 0.603 | 0.656 |

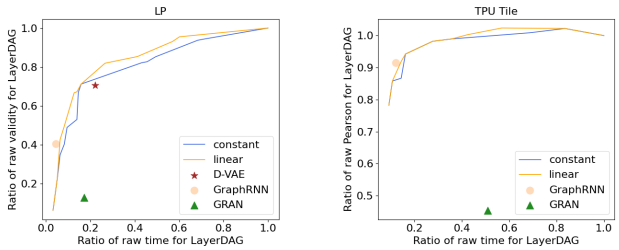


Fig. 4. Quality-efficiency trade-off. $x$-axis is the time budget divided by the raw generation time of LayerDAG. $y$-axis is the generation quality metric divided by the raw generation quality metric of LayerDAG.

we set the number of new nodes to generate in GRAN to be the averaged layer size, and extend mixture of Bernoullis into mixture of multinoullis for the set generation of node attributes. We extend all baselines and LayerDAG for conditional generation by augmenting the input data representations with sinusoidal embeddings [35] of the label.

### B. Evaluation for generation quality (*Q1*)

Table II presents the evaluation results. LayerDAG consistently outperforms all baselines across datasets, where the largest margin is for the LP dataset with hard logical rules. The experiment results demonstrate the aggregated benefit of permutation invariance and multiple rounds of refinement.

### C. Evaluation for layer-index-based denoising schedule (*Q2*)

Figure 4 presents the evaluation of layer-index-based denoising schedule in quality-efficiency trade-off on LP and TPU Tile. For an ablation study, we also experiment with a constant denoising schedule that simply utilizes a constant number of denoising steps smaller than the original number for all layers. Both schedules allow an effective quality-efficiency trade-off, while the generalized linear schedule often yield a better generation quality with the same time budget.

For comparison, we also evaluate the generation time of the baselines using the same batch size. When the GPU memory is insufficient, we use the largest feasible batch size instead. We plot all cases whose generation time is shorter than the original generation time of LayerDAG. Using layer-index-based denoising schedule, LayerDAG exhibits a better trade-off than most baselines except GraphRNN.

## VI. CONCLUSION

We propose LayerDAG, a layerwise diffusion model of DAGs. LayerDAG consistently outperforms the existing generative models of DAGs on four synthetic and real-world datasets. Utilizing a number of refinement rounds proportional to layer depth allows a better or comparable quality-efficiency trade-off compared with the baselines.


## ACKNOWLEDGEMENTS

We thank Haoyu Wang and Yinan Huang for their insights on dataset usage.


## REFERENCES


[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.

[2] S. An, H. Lee, J. Jo, S. Lee, and S. J. Hwang, "Diffusionnag: Predictor-guided neural architecture generation with diffusion models," in *International Conference on Learning Representations*, 2023.

[3] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg, "Structured denoising diffusion models in discrete state-spaces," *Advances in Neural Information Processing Systems*, pp. 17 981–17 993, 2021.

[4] R. Baghdadi, M. Merouani, M.-H. Leghettas, K. Abdous, T. Arbaoui, K. Benatchba *et al.*, "A deep learning based cost model for automatic code optimization," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 181–193, 2021.

[5] M. Burtscher, I. Ganusov, S. Jackson, J. Ke, P. Ratanaworabhan, and N. Sam, "The vpc trace-compression algorithms," *IEEE Transactions on Computers*, vol. 54, no. 11, pp. 1329–1344, 2005.

[6] T. Chen, L. Zheng, E. Yan, Z. Jiang, T. Moreau, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Learning to optimize tensor programs," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

[8] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *International Conference on Learning Representations*, 2020.

[9] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "CktGNN: Circuit graph neural network for electronic design automation," in *International Conference on Learning Representations*, 2023.

[10] C. Dubach, J. Cavazos, B. Franke, G. Fursin, M. F. O'Boyle, and O. Temam, "Fast compiler optimisation evaluation using code-feature based performance prediction," in *Proceedings of the 4th international conference on Computing frontiers*, 2007, pp. 131–142.

[11] C. Hua, S. Luan, M. Xu, Z. Ying, J. Fu, S. Ermon, and D. Precup, "Mudiff: Unified diffusion for complete molecule generation," in *Proceedings of the Second Learning on Graphs Conference*, 2024, pp. 33:1–33:26.

[12] Q. Huang, C. Hong, J. Wawrzynek, M. Subedar, and Y. S. Shao, "Learning a continuous and reconstructible latent space for hardware accelerator design," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 277–287.

[13] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with roc," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 187–198, 2020.

[14] J. Jo, S. Lee, and S. J. Hwang, "Score-based generative modeling of graphs via the system of stochastic differential equations," in *Proceedings of the 39th International Conference on Machine Learning*, 2022, pp. 10 362–10 383.

[15] Z. Jorgensen, T. Yu, and G. Cormode, "Publishing attributed social graphs with formal privacy guarantees," in *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2016, pp. 107–122.

[16] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 622–636.

[17] S. Kaufman, P. Phothilimthana, Y. Zhou, C. Mendis, S. Roy, A. Sabne, and M. Burrows, "A learned performance model for tensor processing units," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 387–400, 2021.

[18] M. Li, J. Y. Liu, L. Sigal, and R. Liao, "GraphPNAS: Learning probabilistic graph generators for neural architecture search," *Transactions on Machine Learning Research*, 2023.

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

[20] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[21] M. Liang, W. Fu, L. Feng, Z. Lin, P. Panakanti, S. Zheng, S. Sridharan, and C. Delimitrou, "Mystique: Enabling accurate and scalable generation of production ai benchmarks," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[22] 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," in *Advances in Neural Information Processing Systems*, 2019.

[23] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *International Conference on machine learning*. PMLR, 2019, pp. 4505–4515.

[24] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 8162–8171.

[25] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon, "Permutation invariant graph generation via score-based generative modeling," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, 2020, pp. 4474–4484.

[26] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 4195–4205.

[27] P. M. Phothilimthana, S. Abu-El-Haija, K. Cao, B. Fatemi, M. Burrows, C. Mendis, and B. Perozzi, "TpuGraphs: A performance prediction dataset on large tensor computational graphs," in *Advances in Neural Information Processing Systems*, 2023.

[28] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 10 684–10 695.

[29] A. Samajdar, J. M. Joseph, M. Denton, and T. Krishna, "Architect: Learning custom architecture design and mapping space," 2021. [Online]. Available: https://arxiv.org/abs/2108.08295

[30] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1715–1725.

[31] S. S. Skiena, *The Algorithm Design Manual*. Springer, 2008.

[32] S. Sridharan, T. Heo, L. Feng, Z. Wang, M. Bergeron, W. Fu, S. Zheng, B. Coutinho, S. Rashidi, C. Man, and T. Krishna, "Chakra: Advancing performance benchmarking and co-design using standardized execution traces," *arXiv preprint arXiv:2305.14516*, 2023.

[33] B. Steiner, C. Cummins, H. He, and H. Leather, "Value learning for throughput optimization of deep learning workloads," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 323–334, 2021.

[34] O. Sỳkora, P. M. Phothilimthana, C. Mendis, and A. Yazdanbakhsh, "Granite: A graph neural network model for basic block throughput estimation," in *2022 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2022, pp. 14–26.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[36] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard, "Digress: Discrete denoising diffusion for graph generation," in *International Conference on Learning Representations*, 2023.

[37] C. Vignac, N. Osman, L. Toni, and P. Frossard, "Midi: Mixed graph and 3d denoising diffusion for molecule generation," in *Machine Learning and Knowledge Discovery in Databases: Research Track*, 2023, pp. 560–576.

[38] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," in *European Conference on Computer Vision*, 2020, pp. 660–676.

[39] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, "High-level synthesis performance prediction using gnns: benchmarking, modeling, and advancing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022, p. 49–54.

[40] M. Yoon, Y. Wu, J. Palowitch, B. Perozzi, and R. Salakhutdinov, "Graph generative model for benchmarking graph neural networks," in *Proceedings of the 40th International Conference on Machine Learning*, 2023, pp. 40 175–40 198.

[41] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 5708–5717.

[42] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, p. 81–93.

[43] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-VAE: A variational autoencoder for directed acyclic graphs," in *Advances in Neural Information Processing Systems*, 2019.

[44] T. Zhang, Z. He, and R. B. Lee, "Privacy-preserving machine learning through data obfuscation," *arXiv preprint arXiv:1807.01860*, 2018.

[45] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen *et al.*, "Ansor: Generating {High-Performance} tensor programs for deep learning," in *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, 2020, pp. 863–879.