LayerDAG: A Layerwise Autoregressive Diffusion Model for Directed Acyclic Graph Generation

Mufei Li[†], **Viraj Shitole**[†], **Eli Chien**[†], **Changhai Man**[†],
Zhaodong Wang[‡], **Srinivas Sridharan**[§], **Ying Zhang**[‡], **Tushar Krishna**[¶], **Pan Li**[†]

[†]Georgia Institute of Technology,
{mufei.li, viraj2, ichien6, cman8, panli}@gatech.edu
[‡]Meta, {zhaodongwang, zhangying}@meta.com
[§]NVIDIA, srisridharan@nvidia.com
[¶]Georgia Institute of Technology, tushar@ece.gatech.edu

Abstract

Directed acyclic graphs (DAGs) are crucial in hardware synthesis and compiler optimization. Synthetic DAGs can be used for benchmarking computing systems while preserving intellectual property. However, DAG generation is challenging due to the inherent directional and logical dependencies. This paper introduces LayerDAG, an autoregressive diffusion model. LayerDAG decouples the strong dependencies into units that can be processed sequentially. By interpreting the partial order of nodes as a sequence of bipartite graphs, LayerDAG leverages autoregressive generation to model directional dependencies and employs diffusion models to capture logical dependencies within each bipartite graph. Experiments demonstrate that LayerDAG outperforms existing DAG generative models, particularly for generating large-scale real-world DAGs with up to 400 nodes. Our implementation is available at https://github.com/Graph-COM/LayerDAG.

1 Introduction

A Directed Acyclic Graph (DAG) represents the order of element sets (Bang-Jensen & Gutin, 2008). Unlike sequences, DAGs incorporate branching and merging, allowing the modeling of complex dependencies and hierarchies. This makes DAGs ideal for representing diverse problem domains such as workload behavior during system execution (Sridharan et al., 2023), operator dependence for program analysis (Phothilimthana et al., 2023; Luo et al., 2021; Cortez et al., 2017), dataflows in circuits (Dong et al., 2023), and cause-effect relationships (Pearl, 1995; Tennant et al., 2020).

Despite the benefits, it is challenging to collect large-scale real-world DAG datasets. For example, execution DAGs can help engineers gain valuable insights into the performance of candidate systems (Luo et al., 2021; Cortez et al., 2017) to optimize the infrastructure accordingly. However, collecting a workload execution DAG (Sridharan et al., 2023) for even a single LLM training job, potentially involving trillions of operations, is extremely prohibitive given the size of modern AI platforms. Even if such a large workload execution DAG could be collected, practical constraints such as the storage requirements of the DAGs and the potential to leak private information about the AI model architecture or the training platform configuration further limit DAG sharing.

DAG generative models may address several challenges. First, a generated representative small DAG can be much more efficient than a real DAG with repetitive patterns. Second, synthetic DAGs enable data sharing without harming intellectual property (Gao et al., 2024; Lin et al., 2020), thereby promoting infrastructure optimization among different parties (Sridharan et al., 2023). Third, a conditional generative model can perform efficient optimization for circuit design (Takagi, 1999; Dong et al., 2023) and compiler optimization (Aho et al., 2006; Phothilimthana et al., 2023).

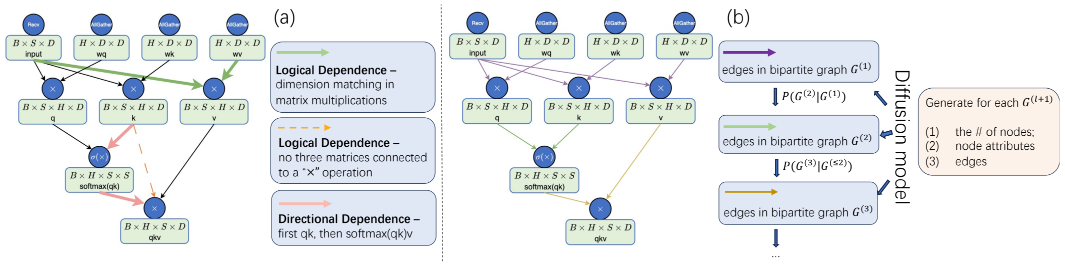
(a)
Recv
AllGather
$B \times S \times D$ input
$H \times D \times D$ wq
$H \times D \times D$ wk
$H \times D \times D$ wv
$B \times S \times H \times D$ q
$B \times S \times H \times D$ k
$B \times S \times H \times D$ v
$\sigma(\times)$
$B \times H \times S \times S$ softmax(qk)
$B \times H \times S \times D$ qkv
Logical Dependence – dimension matching in matrix multiplications
Logical Dependence – no three matrices connected to a "×" operation
Directional Dependence – first qk, then softmax(qk)v
(b)
edges in bipartite graph $G^{(1)}$
$P(G^{(2)}|G^{(1)})$
edges in bipartite graph $G^{(2)}$
$P(G^{(3)}|G^{(\leq 2)})$
edges in bipartite graph $G^{(3)}$
Diffusion model
Generate for each $G^{(l+1)}$
(1) the # of nodes;
(2) node attributes
(3) edges

Figure 1: (a) The computation flow for a transformer layer (Vaswani et al., 2017) encompasses complex logical and directional dependencies. Examples of logical dependencies include 1) dimension matching in matrix multiplications and 2) exactly two matrices pointed to a $\times$ operation. One example of directional dependencies here is SOFTMAX(QK)V being computed after QK. (b) Each DAG has a unique layerwise partition, an ordered partition of nodes/edges into a sequence of bipartite graphs. In LayerDAG, each bipartite graph $G^{(l+1)}$ is generated by a diffusion model conditioned on $G^{(\leq l)}$. LayerDAG generates in order the number of new nodes, their attributes, and the new edges.

DAGs pose significant challenges for developing generative models due to their intrinsic strong directional and logical dependencies, such as control flows, logic gates, and dimension requirements of matrix operations (as illustrated in Fig. 1 (a)). These complexities are further magnified in large-scale DAGs, presenting a unique combination of challenges for both scale and logical rules.

This work proposes the use of autoregressive diffusion models to generate DAGs, aiming to decouple the strong node dependencies in DAGs into manageable units and handle them sequentially. Our model, named LayerDAG, is based on a novel perspective of DAGs: as illustrated in Fig. 1 (b), the partial order of nodes dictated by the DAG structure can be decoupled as a sequence of tokens, each corresponding to a bipartite graph. This perspective enables the natural modeling of directional dependencies in DAGs through autoregression. We further embed diffusion models (Rombach et al., 2022; Vignac et al., 2023) into each autoregressive step to address the logical dependencies.

Our model advances existing models in multiple aspects (Zhang et al., 2019; Li et al., 2023a; An et al., 2023). Although autoregressive models have been adopted by D-VAE (Zhang et al., 2019) and GraphPNAS (Li et al., 2023a) for DAG, they treat either a single node or a node set of constant size as a token. This imposes an order between nodes that should be incomparable, violating the inductive bias of DAGs. Diffusion models have been used to generate undirected graphs (Niu et al., 2020; Jo et al., 2022; Vignac et al., 2023), but they ignore the directional information, while our work demonstrates the necessity of the autoregressive component in modeling DAG directional dependencies. Furthermore, existing works (Zhang et al., 2019; Li et al., 2023a; An et al., 2023) focus on small DAGs (with $\#$nodes $\leq$ 24) for NAS, while our model can generate much larger graphs (up to $\sim$ 400 nodes) for system/hardware benchmarking. Overall, our work is the first to use autoregressive diffusion models for DAG generation, taking the advantages of both autoregressive models and diffusion models to model the strong dependencies commonly in the DAG data.

To assess the model's ability to learn strong directional and logical rules, we construct a challenging synthetic dataset. Additionally, we employ three real-world datasets— computational graphs on Tensor Processing Units (TPU), flow graphs on Field Programmable Gate Arrays (FPGA), and neural architectures deployed on edge device. Each dataset contains thousands of DAGs, and a DAG comprises up to hundreds of nodes. We compare the validity and statistical properties of the synthetic DAGs generated by our model with baselines. To measure benchmarking performance, we use the synthetic labeled DAGs to train surrogate machine learning models to predict TPU runtime, FPGA resource usage, and the inference latency of neural architectures for the three application scenarios. These trained surrogate models are then applied to the real-world DAGs for testing. We compare the predictions given by the surrogate models with the ground-truth behavior of the DAGs on the corresponding systems. Results show that the surrogate models trained on our generated synthetic DAGs consistently outperform the ones derived with baseline generative models.

2 Preliminaries and background

DAG generation A DAG is a directed graph without cycles $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V} = \{1, \cdots, N\}$ is the node set, $\mathcal{E} = \{\cdots, (u, v), \cdots\}$ is the set of directed edges, $\mathbf{X}$ is the node attribute matrix. The edge list $\mathcal{E}$ can also be equivalently represented as the adjacency matrix $\mathbf{A}$. From a collection of DAGs $\{G_i\}_i \sim \mathbb{P}_{\mathcal{G}}$, we aim to learn $\mathbb{P}_{\mathcal{G}}$ with a generative model $\mathbb{P}_{\mathcal{G}}^{\theta}$, where θ denotes model parameters. Additionally, many real-world applications like system benchmarking involve labeled DAGs $\{(G_i, y_i)\}_i$. DAG generation conditioning on these labels is also of interest.

3 Methodology

In this section, we present the LayerDAG framework. We first describe our novel view of DAGs, which uniquely transforms a DAG into a sequence of bipartite graphs. This naturally leads to a layerwise tokenization for autoregressive generation, which tackles the directional dependencies in DAGs. In each autoregressive step, we introduce a layerwise diffusion model that is capable of modeling the dependencies between nodes that are incomparable (no by-default order). We also discuss why our way of DAG decomposition is crucial for model generalization.

3.1 A unique sequential view of DAGs

A DAG establishes a partial order among its nodes based on reachability through directed paths. A partially ordered set can be transformed into an ordered sequence while satisfying the partial order, known as a linear extension or a topological ordering in the context of DAGs (Kahn, 1962; Tarjan, 1972). Previous competitive autoregressive models of DAGs utilize this sequential nature to generate nodes one at a time following a topological ordering (Zhang et al., 2019). However, topological orderings are not unique. This may compromise the model's effectiveness and efficiency. We leave a dedicated discussion about the issue to Section 3.3. On the other hand, non-autoregressive models like diffusion models may be sub-optimal for not explicitly modeling the sequential nature of DAGs.

We map a DAG into a unique sequence by extending topological orderings from a sequence of singleton subsets to general subsets. A DAG is guaranteed to have source nodes, 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 (b)). We refer to $\mathcal{V}^{(1)}, \mathcal{V}^{(2)}, \cdots, \mathcal{V}^{(L)}$ as layers and L as the number of layers. For each layer depth $1 \leq l \leq L-1$, we also take $\mathcal{E}^{(l+1)} = \{(u, v) \in \mathcal{E} | u \in \mathcal{V}^{(\leq l)}, v \in \mathcal{V}^{(l+1)}\}$, and $(\mathcal{E}^{(2)}, \mathcal{E}^{(3)}, \cdots, \mathcal{E}^{(L)})$ forms an ordered partition of $\mathcal{E}$. This layerwise partition naturally extends to arbitrary node and edge attributes. Furthermore, this construction is unique and allows reconstructing the original DAG from the sequence $\left(\mathcal{V}^{(1)}, (\mathcal{V}^{(\leq 1)} \cup \mathcal{V}^{(2)}, \mathcal{E}^{(2)}), \cdots, (\mathcal{V}^{(\leq L-1)} \cup \mathcal{V}^{(L)}, \mathcal{E}^{(L)})\right)$, where $(\mathcal{V}^{(\leq l)} \cup \mathcal{V}^{(l+1)}, \mathcal{E}^{(l+1)})$ is a bipartite graph whose edges point from one part to another part. Such an invertible process of converting a raw data sample into a sequence is known as tokenization, with notable examples being subwords for language models (Sennrich et al., 2016) and patches for image generation (Peebles & Xie, 2023). Essentially, layerwise partition leads to a layerwise tokenization for DAGs.

3.2 LayerDAG

Autoregressive generation The layerwise tokenization motivates the following factorization for a probability distribution $\mathbb{P}(G)$ of DAGs. Let $G = (\mathcal{V}, \mathbf{X}, \mathbf{A})$ be a DAG. In analogy to $\mathcal{V}^{(l)}$ and $\mathcal{V}^{(\leq l)}$, we define $\mathbf{X}^{(l)}$, $\mathbf{X}^{(\leq l)}$, $\mathbf{A}^{(l)}$, and $\mathbf{A}^{(\leq l)}$. In addition, we define $G^{(l)} = \left(\mathcal{V}^{(l)}, \mathbf{X}^{(l)}, \mathbf{A}^{(l)}\right)$ and $G^{(\leq l)} = \left(\mathcal{V}^{(\leq l)}, \mathbf{X}^{(\leq l)}, \mathbf{A}^{(\leq l)}\right)$. Then, we have $\mathbb{P}(G) = \prod_{l=0}^{L-1} \mathbb{P}\left(G^{(l+1)} \;\middle|\; G^{(\leq l)}\right) = \prod_{l=0}^{L-1} \mathbb{P}\left(|\mathcal{V}^{(l+1)}| \;\middle|\; G^{(\leq l)}\right) \mathbb{P}\left(\mathbf{X}^{(l+1)} \;\middle|\; G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right) \mathbb{P}\left(\mathbf{A}^{(l+1)} \;\middle|\; G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$. This factorization naturally leads to a layerwise autoregressive generation framework. Iteratively, 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 $p_\theta\left(\mathbf{A}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$. The generation process terminates when $|\mathcal{V}^{(l+1)}| = 0$.

Layerwise diffusion-based generation To preserve the uniqueness of the layerwise tokenization, we need to generate the set of node attributes and edges in the bipartite graph $G^{(l+1)}$ as a whole by modeling $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$ and $p_\theta\left(\mathbf{A}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$ as set-level generations. This requires capturing complex dependencies between the nodes attributes and edges in the sets, which is crucial for the applications like system and circuit design, where strict logical rules are prevalent, and rule violations can accumulate and propagate over layers of generation. To tackle this issue, we adopt diffusion models for multiple rounds of set refinement in the generation of $G^{(l+1)}$. We employ two separate diffusion processes for node attribute $\mathbf{X}^{(l+1)}$ generation and structure $\mathbf{A}^{(l+1)}$ generation. For simplicity, we focus on categorical node attributes in this work, which find abundant real-world applications like operator types in computational graphs, and thus adopt D3PM (Austin et al., 2021). For more details on layerwise diffusion, see Appendix A.

Implementation For different l's, all modules $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$, $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$, and $p_\theta\left(\mathbf{A}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$ share the same parameter θ, which involves a DAG encoder. We use a bidirectional message passing neural network (BiMPNN) (Wen et al., 2020). A single BiMPNN layer updates node representations with message passing over both the directed edges and their reversed counterparts: $\sigma\left(\mathbf{AHW}_{\text{forward}} + \mathbf{A}^\top\mathbf{HW}_{\text{reverse}} + \mathbf{HW}_{\text{self}}\right)$, where σ is a non-linear layer, $\mathbf{H}$ is the node representation matrix, and $\mathbf{W}$'s are learnable weight matrices. Both layer size generation $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$ and node attribute generation $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$ involve computing graph representations with a set pooling operator over the updated node representations. Let $\mathbf{X}^{(l+1,t)}$ be the node attributes sampled for the $(l+1)^{\text{th}}$ layer at the step t in the reverse node attribute diffusion process. After encoding $(G^{(\leq l)}, t)$ into a context vector $\mathbf{h}_t^{(\leq l)}$, the denoising network ϕ_{θ_X} augments the representations of $\mathbf{X}^{(l+1,t)}$ by $\mathbf{h}_t^{(\leq l)}$ and then applies a transformer without positional encodings (Vaswani et al., 2017) over them for predicting the set of $\mathbf{X}^{(l+1)}$. Similarly, the denoising network ϕ_{θ_A} for edge diffusion augments $(G^{(\leq l)}, t)$ by $(\mathbf{X}^{(l+1)}, \mathbf{A}^{(l+1,t)})$ for computing the node representations of $\mathcal{V}^{(\leq l+1)}$. To predict the probability of edge (u, v) for $u \in \mathcal{V}^{(\leq l)}$ and $v \in \mathcal{V}^{(l+1)}$, it concatenates and transforms the representations of node u, node v, and t with an MLP.

Training aims to maximize the log-likelihood of observed graphs. We decompose this objective to train the three modules independently. To account for the autoregressive nature, we train all modules with teacher forcing. In each training iteration, we randomly sample a real partial DAG up to l layers and train the model to predict the $l+1$-th layer. For a given partial DAG, the training of $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$ is analogous to the standard supervised training of a graph classification model. The training of $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$ and $p_\theta\left(\mathbf{A}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$ follows the established practices for discrete diffusion models.

Sampling The sampling process follows an autoregressive approach. Starting with an empty graph $G^{(\leq 0)}$, we iteratively sample the number of nodes $|\mathcal{V}^{(l+1)}|$ for each subsequent layer using $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$. If $|\mathcal{V}^{(l+1)}|$ is non-zero, we sequentially generate node attributes $\mathbf{X}^{(l+1)}$ and edges $\mathbf{A}^{(l+1)}$ using their respective reverse diffusion processes. We then update $G^{(\leq l+1)}$ with the new nodes, attributes, and edges. This process continues until we predict $|V^{(l+1)}| = 0$.

Conditional generation Given a labeled DAG (G, y), we train LayerDAG to learn the conditional distribution $\mathbb{P}(G|Y)$ by integrating the sinusoidal embeddings of y into the representations of $\mathbf{X}^{(\leq l)}$ for any l. Once trained, the model can generate DAGs conditioned on specified target properties.

3.3 Permutation invariance and model generalization

A critical issue of probabilistic graph generative models is permutation invariance, whether the probability for a model to generate a graph is invariant to the particular node ordering. Non-permutation invariant models, such as autoregressive models that generate one node at a time, require data augmentations with random node orderings during training (You et al., 2018; Li et al., 2018). To sufficiently train the model, one has to enumerate a large number of orderings, which could be exponential in N. LayerDAG is permutation invariant with the aforementioned implementation, potentially holding a good generalization capability with limited computational resources for training.

Proposition 3.1 (permutation invariance of LayerDAG) *For any depth l, $p_\theta\left(|\mathcal{V}^{(l+1)}| \mid G^{(\leq l)}\right)$, $p_\theta\left(\mathbf{X}^{(l+1)} \mid G^{(\leq l)}, |\mathcal{V}^{(l+1)}|\right)$, and $p_\theta\left(\mathbf{A}^{(l+1)} \mid G^{(\leq l)}, \mathbf{X}^{(l+1)}\right)$ are permutation invariant. Hence, LayerDAG is permutation invariant.*

MPNNs are permutation equivariant, and pooling operators are permutation invariant. It follows that for any permutation Π, we have $\mathbb{P}\left(\Pi^{(l+1)}(G^{(l+1)}) \mid \Pi^{(\leq l)}(G^{(\leq l)})\right) = \mathbb{P}\left(G^{(l+1)} \mid G^{(\leq l)}\right)$, where $\Pi^{(l+1)}$ corresponds to $\mathcal{V}^{(l+1)}$ and $\Pi^{(\leq l)}$ corresponds to $\mathcal{V}^{(\leq l)}$. Finally, $\mathbb{P}(G) = \prod_{l=0}^{L-1} \mathbb{P}\left(G^{(l+1)} \mid G^{(\leq l)}\right)$ is permutation invariant.

4 Experiments

The end applications of DAG generative models pose multifaceted requirements that motivate our empirical studies. **Q1**) Real-world DAGs encompass complex logical and directional dependencies, where violations can directly invalidate generated samples. How effectively can LayerDAG capture these dependencies by learning from valid DAGs? **Q2**) Beyond validity, the application of synthetic data for system and hardware benchmarking necessitates the preservation of correlations between DAGs and system metrics such as throughput, latency, and memory footprint. How effectively can LayerDAG perform DAG generation conditioning on these metrics to meet these requirements?

Baselines We consider three non-diffusion autoregressive baselines – **GraphRNN**, **D-VAE**, and **GraphPNAS**. GraphRNN originally tackles undirected graph structure generation (You et al., 2018), and we adopt an extension of it for attributed DAG generation Zhang et al. (2019). D-VAE is a variational auto-encoder and employs a nodewise autoregressive decoder. Both GraphRNN and D-VAE employ topological orderings and one-hot encoding of node IDs (Zhang et al., 2019). GraphPNAS sequentially generates incident edges for a node set of constant size, using a mixture of Bernoulli distributions to model intra-set dependencies. To compare the capability of LayerDAG in set generation against GraphPNAS, we further adapt GraphPNAS by using mixtures of multinoulli distributions for generating a set of node attributes and setting the node set size to be the averaged layer size. For pure diffusion baselines, we implement **OneShotDAG**, a non-autoregressive variant of LayerDAG. For ablation study of multiple rounds of refinement, we report results with a single denoising step, denoted by **LayerDAG ($\mathbf{T} = \mathbf{1}$)**. For details of model extensions, see Appendix B.

4.1 Generating synthetic DAGs with strong logical rules (**Q1**)

To evaluate the model's capability in capturing logical rules, we propose a synthetic dataset of latent preferential DAGs (**LP**). LP adheres to various hard constraints, including a constraint on the balanced level of binary node attributes among the node's predecessors. Specifically, we enforce that $\frac{\lfloor |n_v^{(0)} - n_v^{(1)}|/2 \rfloor}{(n_v^{(0)} + n_v^{(1)})/2} \leq \rho$ for any node v, where $\lfloor \cdot \rfloor$ is the floor function, and $n_v^{(i)}$ is the number of node v's predecessors with attribute i for $i \in \{0, 1\}$. The parameter $\rho \in \{0, 0.5, 1\}$ helps assess how model performance varies under different degrees of constraint, where a lower value imposes stricter constraints ($\rho = 0$ indicating $|n_v^{(0)} - n_v^{(1)}| \leq 1$). See Appendix C for a full dataset description.

We use LP with different ρ's to generate the datasets D_ρ and train different generative models based on D_ρ. After training, we use each model to generate DAGs of the same number as that in D_ρ. To evaluate the hard logical constraints, we assess the validity of generated DAGs by measuring the proportion of generated DAGs that satisfy the imposed hard constraints. To evaluate the soft constraints, we compare the distributions of graph statistics between generated DAGs and an equal number of real DAGs. We measure the 1-Wasserstein distance (W_1) between the distributions of layer numbers (L) for the two graph sets. Additionally, we measure Maximum Mean Discrepancy (MMD) You et al. (2018) between two sets of graph statistic distributions corresponding to the graph sets. Specifically, we report MMD for the distributions of layer size ($|\mathcal{V}^{(l)}|$).

Table 1 presents the evaluation. LayerDAG consistently outperforms all the other models in terms of validity, with substantial margins observed under stricter logical rules (lower ρ values), about 20% in absolute value. Nodewise autoregressive models (GraphRNN and D-VAE) that directly encode node IDs struggle to learn strict logical rules. Meanwhile, a mixture of Bernoulli/multinoulli distribution is also not expressive enough, resulting in GraphPNAS achieving low validity scores. The ablation studies against the non-autoregressive variant (OneShotDAG) and the single-denoising-step variant

Table 1: Evaluation results on LP. Best results are in **bold**.

Model	$\rho = 0$			$\rho = 0.5$			$\rho = 1$		
	Validity ↑	W_1 / MMD ↓		Validity ↑	W_1 / MMD ↓		Validity ↑	W_1 / MMD ↓	
		$L(\times 10^{-1})$	$\lvert\mathcal{V}^{(l)}\rvert(\times 10^{-1})$		L	$\lvert\mathcal{V}^{(l)}\rvert$		L	$\lvert\mathcal{V}^{(l)}\rvert$
D-VAE	0.27 ± 0.03	8.7 ± 1.0	1.9 ± 0.3	0.37 ± 0.04	9.8 ± 1.6	1.9 ± 0.6	0.89 ± 0.01	8.8 ± 0.9	1.9 ± 0.5
GraphRNN	0.25 ± 0.02	9.8 ± 0.2	1.2 ± 0.2	0.34 ± 0.07	12.0 ± 0.0	1.8 ± 0.2	0.59 ± 0.02	14.0 ± 1.0	2.1 ± 0.1
GraphPNAS	0.23 ± 0.04	17.0 ± 4.0	2.2 ± 0.7	0.24 ± 0.03	20.0 ± 3.0	3.2 ± 1.3	0.67 ± 0.04	10.0 ± 3.0	0.8 ± 0.6
OneShotDAG	0.37 ± 0.02	6.4 ± 0.9	1.5 ± 0.1	0.31 ± 0.07	3.9 ± 0.7	1.3 ± 0.0	0.50 ± 0.08	4.1 ± 2.4	1.1 ± 0.4
LayerDAG ($T = 1$)	0.26 ± 0.06	$\mathbf{1.6 \pm 0.8}$	$\mathbf{0.14 \pm 0.0}$	0.36 ± 0.02	$\mathbf{1.3 \pm 0.3}$	$\mathbf{0.12 \pm 0.1}$	0.95 ± 0.01	$\mathbf{2.0 \pm 0.1}$	$\mathbf{0.08 \pm 0.0}$
LayerDAG	$\mathbf{0.56 \pm 0.02}$	$\mathbf{1.6 \pm 1.0}$	$\mathbf{0.10 \pm 0.0}$	$\mathbf{0.63 \pm 0.00}$	$\mathbf{1.8 \pm 1.1}$	$\mathbf{0.06 \pm 0.0}$	$\mathbf{0.96 \pm 0.02}$	$\mathbf{1.9 \pm 0.6}$	$\mathbf{0.10 \pm 0.3}$

Table 2: Evaluation results for conditional generation. Best results are in **bold**.

Model	TPU Tile				HLS				NA-Edge			
	ML		W_1 / MMD ↓		ML		W_1 / MMD ↓		ML		W_1 / MMD ↓	
	Pearson	MAE	L	$\lvert\mathcal{V}^{(l)}\rvert(\times 10^{-1})$	Pearson	MAE	L	$\lvert\mathcal{V}^{(l)}\rvert(\times 10^{-1})$	Pearson	MAE	$L(\times 10)$	$\lvert\mathcal{V}^{(l)}\rvert(\times 10^{-1})$
Real graphs	0.75 ± 0.01	0.9 ± 0.0			0.98 ± 0.00	0.3 ± 0.0			0.996 ± 0.000	0.3 ± 0.0		
D-VAE	0.50 ± 0.01	1.4 ± 0.0	2.6 ± 0.1	1.3 ± 0.4	0.82 ± 0.04	1.2 ± 0.1	$\mathbf{3.2 \pm 1.7}$	1.5 ± 0.2	0.877 ± 0.026	2.3 ± 0.8	3.6 ± 0.7	7.3 ± 1.3
GraphRNN	0.62 ± 0.02	1.3 ± 0.0	1.9 ± 0.2	0.4 ± 0.1	0.79 ± 0.03	$\mathbf{1.1 \pm 0.1}$	11 ± 1.0	2.4 ± 0.3	0.980 ± 0.010	1.0 ± 0.1	13 ± 2.0	14 ± 4.0
GraphPNAS	0.24 ± 0.10	2.1 ± 0.6	6.2 ± 0.5	1.0 ± 0.3	0.66 ± 0.05	2.5 ± 0.6	26 ± 0.0	6.8 ± 0.1	0.619 ± 0.118	7.7 ± 2.6	15 ± 0.0	14 ± 0.0
OneShotDAG	0.56 ± 0.02	1.4 ± 0.1	6.9 ± 0.2	3.5 ± 0.1	0.73 ± 0.03	1.4 ± 0.1	21 ± 0.0	4.4 ± 0.1	0.887 ± 0.038	3.4 ± 1.0	14 ± 0.0	9.2 ± 0.0
LayerDAG ($T = 1$)	0.37 ± 0.11	2.0 ± 0.4	2.0 ± 0.4	2.1 ± 0.2	0.27 ± 0.26	2.1 ± 0.2	7.9 ± 2.2	5.2 ± 0.2	0.956 ± 0.011	3.1 ± 2.0	6.1 ± 1.6	4.7 ± 4.5
LayerDAG	$\mathbf{0.65 \pm 0.01}$	$\mathbf{1.2 \pm 0.1}$	$\mathbf{1.3 \pm 0.4}$	$\mathbf{0.1 \pm 0.0}$	$\mathbf{0.85 \pm 0.02}$	$\mathbf{1.1 \pm 0.2}$	11 ± 3.0	$\mathbf{1.4 \pm 0.0}$	$\mathbf{0.990 \pm 0.005}$	$\mathbf{0.9 \pm 0.3}$	$\mathbf{1.3 \pm 0.2}$	$\mathbf{0.4 \pm 0.1}$

($T = 1$) underscore the importance of combining autoregressive layerwise generation and diffusion in modeling strong directional and logical rules for DAG generation. In terms of graph statistics, the layerwise autoregressive models, LayerDAG ($T = 1$) and LayerDAG, yield a better performance in capturing layerwise patterns (L and $|\mathcal{V}^{(l)}|$), demonstrating the benefit of layerwise generation.

4.2 Conditional generation for real-world computation graph datasets (**Q2**)

Datasets. We repurpose three real-world DAG property prediction datasets. 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 y indicating the runtime averaged over a set of compilation configurations (Phothilimthana et al., 2023). **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) (Wu et al., 2022). **NA-Edge** is a collection of DAGs representing neural architectures, with labels indicating their inference latency on mobile CPU (Dong & Yang, 2020; Zhang et al., 2021). We perform a random train/val/test split for all datasets. For more details, see Appendix E.

Evaluation. 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 (Wu et al., 2022). 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 (Chen et al., 2018; Jia et al., 2020; Phothilimthana et al., 2023). 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. Then, 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 (Yoon et al., 2023; Li et al., 2023b), we train two ML models with BiMPNN using the same automated pipeline respectively on the real and synthetic training/validation subsets. We then compare the performance of the two models on the real test set. A generative model is considered better if its corresponding model achieves a performance closer to that of the model trained on the real subsets.

Table 2 presents the evaluations. We report two metrics for ML-based evaluation – Pearson correlation coefficient that compares the relative label differences of DAGs in the test set based on the predicted and ground-truth labels, and mean absolute error (MAE) that compares the absolute difference between the predicted and ground-truth labels. LayerDAG consistently achieves the best performance in ML-based evaluation. Furthermore, we assess discrepancies in graph statistics between the real validation subset and a synthetic validation subset via the same metrics used in Sec.4.1, where LayerDAG also achieves the best performance in general. Overall, these evaluation results are aligned with previous observations made for the LP dataset.

5 CONCLUSION

We propose LayerDAG, a layerwise autoregressive diffusion model for DAGs. Extensive experiments on synthetic and real-world datasets demonstrate a superior capability of LayerDAG in modeling the strong dependencies common in DAG data.

ACKNOWLEDGEMENTS

We are deeply grateful to Haoyu Wang and Yinan Huang for their insights on dataset adoption and adaptation. M. Li, V. Shitole, E. Chien and P. Li are partially supported by NSF awards PHY-2117997, IIS-2239565, IIS-2428777, and CCF-2402816; DOE award DE-FOA-0002785; JPMC faculty awards; Microsoft Azure Research Credits for Generative AI; and Meta research award.

REFERENCES

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.

Sohyun An, Hayeon Lee, Jaehyeong Jo, Seanie Lee, and Sung Ju Hwang. Diffusionnag: Predictor-guided neural architecture generation with diffusion models. In *International Conference on Learning Representations*, 2023.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, pp. 17981–17993, 2021.

Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. *Advances in Neural Information Processing Systems*, 31, 2018.

Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 153–167, 2017.

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Zehao Dong, Weidong Cao, Muhan Zhang, Dacheng Tao, Yixin Chen, and Xuan Zhang. CktGNN: Circuit graph neural network for electronic design automation. In *International Conference on Learning Representations*, 2023.

Yubo Gao, Maryam Haghifam, Christina Giannoula, Renbo Tu, Gennady Pekhimenko, and Nandita Vijaykumar. Proteus: Preserving model confidentiality during graph optimizations. In *Proceedings of Machine Learning and Systems*, 2024.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems*, 2:187–198, 2020.

Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 10362–10383, 2022.

A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.

Muchen Li, Jeffrey Yunfan Liu, Leonid Sigal, and Renjie Liao. GraphPNAS: Learning probabilistic graph generators for neural architecture search. *Transactions on Machine Learning Research*, 2023a.

Mufei Li, Eleonora Kreačić, Vamsi K. Potluru, and Pan Li. Graphmaker: Can diffusion models generate large attributed graphs? *arXiv preprint arXiv:2310.13833*, 2023b.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pp. 464–483, 2020.

Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 412–426, 2021.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 8162–8171, 2021.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 4474–4484, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, 2023.

Phitchaya Mangpo Phothilimthana, Sami Abu-El-Haija, Kaidi Cao, Bahare Fatemi, Michael Burrows, Charith Mendis, and Bryan Perozzi. TpuGraphs: A performance prediction dataset on large tensor computational graphs. In *Advances in Neural Information Processing Systems*, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022.

Rico Sennrich, Barry Haddow, and Alexandra 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)*, pp. 1715–1725, 2016.

Srinivas Sridharan, Taekyung Heo, Louis Feng, Zhaodong Wang, Matt Bergeron, Wenyin Fu, Shengbao Zheng, Brian Coutinho, Saeed Rashidi, Changhai Man, and Tushar Krishna. Chakra: Advancing performance benchmarking and co-design using standardized execution traces. *arXiv preprint arXiv:2305.14516*, 2023.

Kazuyoshi Takagi. Design and analysis of vlsi circuits based on directed acyclic graphs. 1999.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2): 146–160, 1972.

Peter W G Tennant, Eleanor J Murray, Kellyn F Arnold, Laurie Berrie, Matthew P Fox, Sarah C Gadd, Wendy J Harrison, Claire Keeble, Lynsie R Ranker, Johannes Textor, Georgia D Tomova, Mark S Gilthorpe, and George T H Ellison. Use of directed acyclic graphs (DAGs) to identify confounders in applied health research: review and recommendations. *International Journal of Epidemiology*, 50(2):620–632, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2023.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pp. 660–676, 2020.

Nan Wu, Hang Yang, Yuan Xie, Pan Li, and Cong Hao. High-level synthesis performance prediction using gnns: benchmarking, modeling, and advancing. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, pp. 49–54, 2022.

Minji Yoon, Yue Wu, John Palowitch, Bryan Perozzi, and Russ Salakhutdinov. Graph generative model for benchmarking graph neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 40175–40198, 2023.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5708–5717, 2018.

Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin 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*, pp. 81–93, 2021.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-VAE: A variational autoencoder for directed acyclic graphs. In *Advances in Neural Information Processing Systems*, 2019.

A DETAILS ON LAYERWISE DIFFUSION

Discrete denoising diffusion We adopt D3PM (Austin et al., 2021) for generative diffusion of discrete data. D3PM has two phases, a forward and a reverse process. Let each entry of $\mathbf{Z}^{(0)} \in \{0,1\}^{M\times C}$ be a one-hot encoding of a categorical attribute with C possible values. The forward process uses T consecutive steps to progressively corrupt $\mathbf{Z}^{(0)} \to \mathbf{Z}^{(1)} \to \cdots$ into purely random variables $\mathbf{Z}^{(T)}$, where $\mathbf{Z}^{(T)}$ can be drawn from a prior categorical distribution. 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. 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 $\overline{\mathbf{Q}}^{(t+1)} = \mathbf{Q}^{(1)}\mathbf{Q}^{(2)}\cdots\mathbf{Q}^{(t+1)}$, which allows parallel training across samples and time steps. An instantiation of $\{\overline{\mathbf{Q}}^{(t)}\}_t$ is valid as long as $\lim_{t\to T}\overline{\mathbf{Q}}^{(t)}$ is a known prior distribution.

A denoising network ϕ_θ is trained to predict the uncorrupted data $\mathbf{Z}^{(0)}$ from $(\mathbf{Z}^{(t)}, t)$. During the reverse generation process, the trained denoising network is used to convert $\mathbf{Z}^{(T)}$ drawn from the

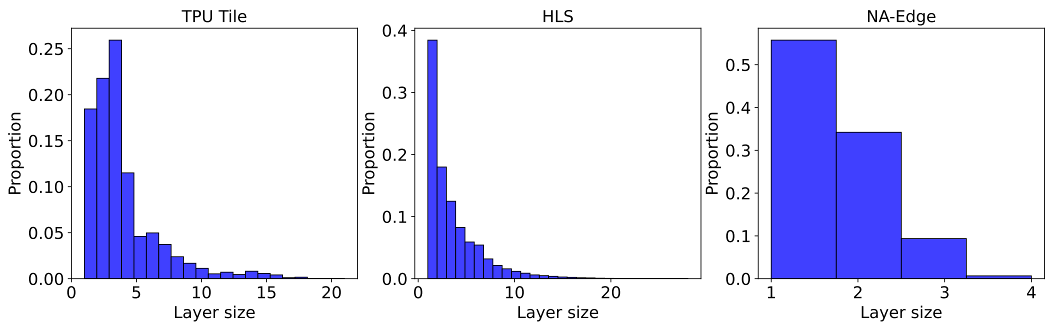

Figure 2: Layer size distribution in the real-world datasets.

prior distribution into realistic data. Iteratively, we compute and sample from $p_\theta(\mathbf{Z}^{(t-1)}|\mathbf{Z}^{(t)}, t) \propto \mathbf{Z}^{(t)}(\mathbf{Q}^{(t)})^\top \odot \hat{\mathbf{Z}}^{(0)}\overline{\mathbf{Q}}^{(t-1)}$, where $^\top$ denotes transpose, and $\odot$ denotes element-wise product.

Following the practice of DiGress, we use the empirical marginal distribution of a categorical attribute $\mathbf{m} \in \mathbb{R}^C$ as its corresponding prior distribution, which was observed to yield more efficient generation compared with a uniform prior in practice. 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 (Nichol & Dhariwal, 2021), $\mathbf{I} \in \mathbb{R}^{C\times C}$ is the identity matrix, and $\mathbf{1} \in \mathbb{R}^C$ is the one-valued vector. As $t \rightarrow T$, the probability for real categorical attributes to be corrupted into random samples from the prior distribution approaches 1.

In addition, we propose two modifications to the diffusion process that better preserve the layerwise patterns of DAGs. To handle the potential uneven graph sparsity with respect to layer depth l, we set the prior probability of a directed edge (u, v) for $u \in \mathcal{V}^{(\leq l)}$ and $v \in \mathcal{V}^{(l+1)}$ to be $\frac{\min(|\mathcal{V}^{(\leq l)}|, d_{\text{in}})}{|\mathcal{V}^{(\leq l)}|}$, where d_{in} is the average node in-degree of the training data. As all nodes in $\mathcal{V} \setminus \mathcal{V}^{(1)}$ have at least one predecessor, we enforce this property in the sampling for graph structure corruption and generation.

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 $t = T_X - 1, T_X - 2, \cdots, 0$. Similarly, the edge prediction module iteratively samples $\mathbf{A}^{(l+1,t)}$ with a denoising network $\phi_{\theta_E}\left(G^{(\leq l)}, \mathbf{X}^{(l+1)}, \mathbf{A}^{(l+1,t+1)}, t+1\right)$.

B Baseline extensions

General extension for DAG generation. To extend generative models of undirected graphs for DAG generation, we constrain the structure generation to the lower-triangular part of adjacency matrices with a topological ordering.

GraphRNN. GraphRNN originally employs rows of an adjacency matrix for both the encoder input and decoder output. Following the practice of (Zhang et al., 2019), we augment them with one-hot encodings of categorical attributes for encoding and predicting the node attributes.

GraphPNAS. We use two separate encoders for node attribute prediction and structure prediction as in LayerDAG. Repeatedly, the model first predicts the categorical attributes of a new set of nodes given the partially generated DAG, and then it predicts the incident edges of the new nodes given the partial DAG and the new node attributes. We model the termination of DAG generation as an extra node attribute, which enables the model to generate DAGs with an arbitrary number of nodes different from multiples of the pre-specified size.

We extend the official implementation for D-VAE and GRAN, both use MIT license.

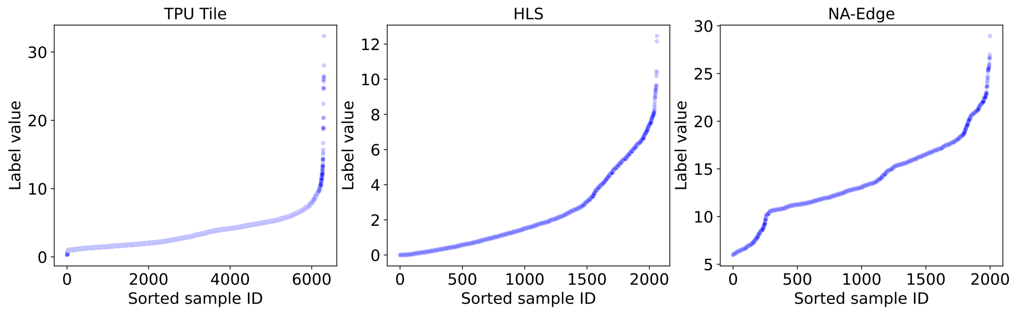

Figure 3: Scatter plot of the sorted labels. The label distributions in the real-world datasets are long-tailed.

C A SYNTHETIC DATASET OF LATENT PREFERENTIAL DAGS (LP)

A latent preferential DAG is constructed as follows.

1. Randomly sample the number of layers $L \sim \mathcal{U}\{2, 5\}$.
2. Create the first layer of $|\mathcal{V}^{(1)}| \sim \mathcal{U}\{1, 5\}$ nodes. Each node is associated with a random binary attribute 0 or 1.
3. For $l = 2, \cdots, L$:
    (a) Create a new layer of $|\mathcal{V}^{(l)}| \sim \mathcal{U}\{1, 5\}$ nodes. Each node is associated with a random binary attribute 0 or 1.
    (b) For each new node v:
        i. An edge (u, v) is created with a probability $\propto \frac{1}{d_v - d_u}$, where $d_v = l$ is the depth of node v and $d_v > d_u$.
        ii. Let $n_v^{(i)}$ be the number of predecessors of node v with attribute i for $i \in \{0, 1\}$. We enforce that $\frac{\lfloor |n_v^{(0)} - n_v^{(1)}|/2 \rfloor}{(n_v^{(0)} + n_v^{(1)})/2} \leq \rho$, where $\lfloor \cdot \rfloor$ is the floor function.
        iii. $n_v = n_v^{(0)} + n_v^{(1)} \sim \mathcal{U}\{1, 4\}$.

The logical rules for predecessors get increasingly relaxed as ρ goes from 0 to 1.

D VARIANT OF THE LP DATASET FOR ASSESSING THE VALIDITY OF GENERATED NODE ATTRIBUTES

We extend the original synthetic dataset with $\rho = 0$ (full balance requirement). Specifically,

- Each node now has three binary attributes (previously one).
- For nodes in the first layer, all three binary attributes are randomly sampled from prior Bernoulli distributions.
- The first attribute is still used for determining edge connections based on balance.
- The second and third attributes of an intermediary node are assigned the most and least common corresponding attribute values among its predecessors, respectively. This design is inspired by real-world scenarios, such as tensor dimension matching in computational graphs. In cases of ties, attribute values are assigned randomly.

In addition to the previously adopted metrics, we also report balance-only validity and feature-only validity as key components of the full validity metric, as well as MMD for feature distributions. Table 3 compares LayerDAG with the two most competitive baselines concluded from the other experiments, with results averaged over three random seeds. Overall, LayerDAG achieves the best performance across all metrics. This demonstrates LayerDAG's superior capability in generating valid attributes and learning the attribute distribution.

Table 3: Evaluation results on the extended synthetic dataset for $\rho = 0$ (full balance requirement). Best results are in **bold**.

Model	Validity ↑			W_1 / MMD ↓		
	balance	attribute	full	L	$\lvert\mathcal{V}^{(l)}\rvert$	attribute
D-VAE	0.536	0.057	0.039	$5.1e^{-1}$	$1.6e^{-1}$	$2.5e^{-3}$
GraphRNN	0.547	0.172	0.094	$5.8e^{-1}$	$1.2e^{-1}$	$8.5e^{-4}$
LayerDAG	**0.578**	**0.274**	**0.195**	$\mathbf{1.8e^{-1}}$	$\mathbf{2.5e^{-3}}$	$\mathbf{7.8e^{-4}}$

Table 4: Dataset statistics. $\lvert\mathcal{V}\rvert$, $\lvert\mathcal{E}\rvert$, and L are averaged over graphs. $\lvert\mathcal{V}^{(l)}\rvert$ 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**	**label info**
TPU Tile	6, 301	40.8	394	42.9	711	11.2	72	3.6	21	1	TPU runtime
HLS	2, 062	88.6	356	110.7	477	27.75	78	3.2	28	7	FPGA resource usage
NA-Edge	2, 000	231.1	339	265.8	385	149.1	185	1.5	4	14	mobile CPU inference latency

E Details on real-world datasets and adaptation of them

Table 4 presents the dataset statistics.

TPU Tile. In the original dataset, each data sample includes a computational graph, a compilation configuration, and the execution time of the graph on TPU when compiled with that configuration. A single graph may appear in multiple data samples and have multiple associated compilation configurations. We simplify the dataset by averaging the runtime across all compilation configurations for each graph. As the labels of the test set were not released, we re-perform a split of the labeled samples.

HLS. We randomly choose 20% of the original graphs.

For all three real-world datasets, we perform a 80/10/10 random split of the dataset. All three datasets exhibit long-tailed layer size distributions (Fig. 2) and label distributions (Fig. 3).

TPU Tile was originally part of the TpuGraphs dataset, which uses Apache-2.0 license. The HLS dataset is not released with a license. NA-Edge is released with an MIT license as part of the nn-Meter project.

F Experiment details

F.1 Model development

For the synthetic LP dataset, we tune the hyperparameters based on validity. For conditional generation, we tune the hyperparameters based on Pearson correlation coefficient. For each experiment, an early stop is performed based on validation accuracies (for layer size prediction) or negative log likelihoods (for diffusion).

F.2 ML-based evaluation

We perform ML-based evaluation by implementing a standardized AutoML pipeline. Based on empirical studies, the best BiMPNN model is selected based on validation Pearson correlation coefficient, and the best Kaggle model is selected based on validation mean absolute error.

F.3 Experiments Compute Resources

We have access to an Azure virtual machine equipped with 2 NVIDIA A100 PCIe GPUs, each with 80 GB of memory, 48 non-multithreaded AMD EPYC Milan processor cores, and 440 GiB of system memory.

F.4 Implementation

Our implementation is based on PyTorch 1.12.0 (Paszke et al., 2019) and DGL 1.1.0 (Wang et al., 2019).