GraDE: A Graph Diffusion Estimator for Frequent Subgraph Discovery in Neural Architectures

Yikang Yang[1,3] **Zhengxin Yang**[1,2*] **Minghao Luo**[4] **Luzhou Peng**[1,3]
Hongxiao Li[1,3] **Wanling Gao**[1,2] **Lei Wang**[1,2] **Jianfeng Zhan**[1,2]
Institute of Computing Technology, Chinese Academy of Sciences[1]
BenchCouncil (International Open Benchmark Council)[2]
University of Chinese Academy of Sciences[3]
Department of Computer Science McCormick School of Engineering
Northwestern University Evanston, IL, USA[4]
{yangyikang23s, yangzhengxin}@ict.ac.cn

Abstract

Finding frequently occurring subgraph patterns or network motifs in neural architectures is crucial for optimizing efficiency, accelerating design, and uncovering structural insights. However, as the subgraph size increases, enumeration-based methods are perfectly accurate but computationally prohibitive, while sampling-based methods are computationally tractable but suffer from a severe decline in discovery capability. To address these challenges, this paper proposes GraDE, a diffusion-guided search framework that ensures both computational feasibility and discovery capability. The key innovation is the Graph Diffusion Estimator (GraDE), which is the first to introduce graph diffusion models to identify frequent subgraphs by scoring their typicality within the learned distribution. Comprehensive experiments demonstrate that the estimator achieves superior ranking accuracy, with up to 114% improvement compared to sampling-based baselines. Benefiting from this, the proposed framework successfully discovers large-scale frequent patterns, achieving up to 30× higher median frequency than sampling-based methods.

1 Introduction

Finding frequent subgraph patterns or network motifs [21, 44] is significant for characterizing the fundamental structural properties of neural architectures. From a computational perspective, these motifs reveal recurring execution patterns that are essential for optimizing efficiency [31, 40]. From a design perspective, the discovery of such patterns provides a modular library of reusable components to accelerate architecture design [16, 17]. From an analytical perspective, these motifs serve as structural fingerprints that uncover the intrinsic regularities governing neural networks [37, 46]. These advantages advance both theoretical understanding and practical engineering of neural networks.

However, finding network motifs is extremely complex, inherently challenged by a trade-off between accuracy and scalability. Enumeration-based methods offer absolute accuracy by exhaustively counting all possible subgraphs of size k [10, 44]. Unfortunately, this process is NP-hard, becoming computationally prohibitive as k increases [27]. Sampling-based methods attempt to ensure scalability by discovering patterns from a sampled subset of subgraphs [19, 25, 28, 39]. Although these methods are computationally tractable, they suffer from a severe decline in discovery capability as k increases. Specifically, as the search space expands exponentially with k, the extreme sparsity of samples leads to high variance and poor representativeness, causing a sharp decline in discovery capability.

*Zhengxin Yang is the corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

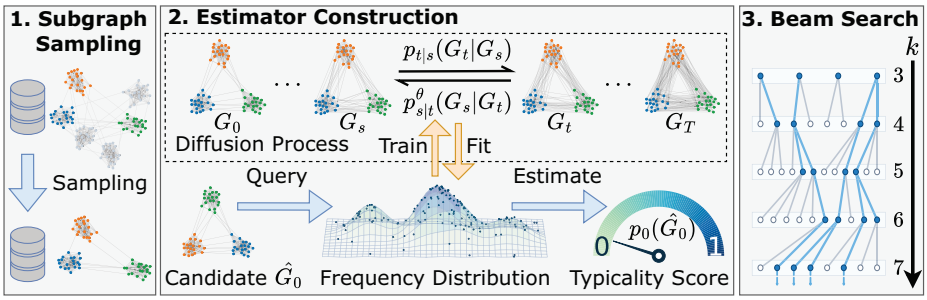


Figure 1: Overview of the GraDE framework. The proposed pipeline operates in three stages: (1) **Subgraph Sampling**, where a representative training set is collected from target architectures; (2) **Estimator Construction**, where a graph diffusion model is trained to fit the subgraph frequency distribution, enabling the GraDE estimator to serve as a frequency surrogate by scoring the structural typicality of any query candidate instance; (3) **Beam Search**, where motifs are discovered by iteratively expanding candidates and leveraging GraDE scores to retain only the top-N promising subgraphs at each size k to prune the search space.

To address these challenges, this paper proposes GraDE, a diffusion-guided search framework to ensure computational feasibility and high discovery capability. The core of this framework is the Graph Diffusion Estimator (GraDE), which is inspired by the capability of generative models in distribution learning [2, 5, 13, 18, 22, 26, 32, 33, 36, 41]. This estimator is built upon the key insight that generative models can be inverted to grade structural typicality of any instance, thereby serving as a reliable surrogate for subgraph frequency. Unlike sampling-based methods that treat each subgraph as an isolated entity, GraDE uses graph diffusion models to capture structural correlations, thereby generalizing from sparse data to a continuous probabilistic landscape.

As shown in Figure 1, the GraDE framework follows a three-stage pipeline: **(1) Subgraph Sampling** collects a representative training set from the architecture distribution; **(2) Estimator Construction** builds the Graph Diffusion Estimator by training a graph diffusion model to capture the structural distribution; **(3) Beam Search** discovers motifs by iteratively expanding candidates and pruning unpromising branches via GraDE scores. By replacing exponential subgraph counting with efficient model inference, GraDE ensures computational feasibility for large-scale patterns.

Comprehensive experiments are conducted on synthesized neural architectures (NAS-Bench-101 [43], NAS-Bench-201 [8], NAS-Bench-301 [47], NAS-Bench-NLP [15]), real-world neural architectures (Younger [42]), and non-neural domains (HighSchool [20] and PROTEINS [3]). The evaluation firstly confirms the effectiveness of the graph diffusion estimator as a subgraph frequency surrogate. GraDE outperforms baselines significantly, especially under sample-sparse conditions, with Spearman's rank correlation coefficient improving by up to 114% on synthesized architectures, 33% on real-world architectures, and 30% on non-neural datasets. When integrated into the search pipeline, GraDE identifies motifs with up to 30× higher median frequency than sampling-based methods for large subgraphs. These results demonstrate the superior discovery capability of the GraDE framework.

The main contributions of this paper are: (1) a novel **subgraph frequency estimator**, GraDE, which is the first to repurpose graph diffusion models for grading structural typicality; (2) a **search framework** that integrates GraDE with a beam search strategy to ensure computational feasibility and high discovery capability; (3) **comprehensive experiments** across various datasets confirming the effectiveness of both the estimator and the framework. The code is available at https://github.com/buaayyk/GraDE.

2 Related Work

2.1 Network Motif Discovery

Network motifs are frequent and representative subgraph patterns in a graph [21]. Traditional discovery methods include two primary paradigms: enumeration-based and sampling-based methods.

Enumeration-based methods, such as ESU [39], count all subgraphs of size k to get exact results. While absolutely precise, these methods suffer from exponential computational complexity. Although

optimizations [24, 29, 38] have been proposed to accelerate execution, they remain fundamentally constrained by search space growth, making exact enumeration intractable for large subgraphs.

Sampling-based methods were introduced to solve these efficiency issues. Many of these approaches add random sampling to the enumeration process to avoid high costs while keeping the results unbiased, such as Rand-ESU [39], Rand-FaSE [25], and Rand-Gries [28]. Other methods sample subgraphs based on local structures [14, 30], such as NRS [19], but these are often biased. However, as k increases, the subgraph space grows exponentially, causing the sampled data to become extremely sparse. This sparsity leads to a significant decline in discovery capability, making these methods unreliable for identifying large-scale motifs in complex neural architecture datasets.

2.2 Graph Generative Models

Graph generative models aim to learn the distribution of graph data to generate new samples with similar structures [9]. Early approaches include GraphVAE [33], which uses variational inference for graph encoding and decoding, and GraphRNN [45], which treats generation as a sequential decision process. Other methods leverage adversarial training, such as NetGAN [2] and MolGAN [5], or utilize invertible mappings in normalizing flow-based models [18].

Recently, diffusion models achieved state-of-the-art graph generation performance. Initial works [22, 32] directly applied Gaussian noise to adjacency matrices, which often destroyed the discrete structural constraints of graphs. To resolve this, DiGress [36] introduced discrete diffusion processes, adding noise to categorical node and edge attributes to better preserve structural fidelity. Based on this, DisCo and other methods [34, 41] extended the process into continuous time, significantly enhancing the flexibility and efficiency of the generation process. More recently, research has shifted toward flow-matching methods [26], which define vector fields to transform noise into target distributions.

Despite their success in generation, these models have not yet been utilized as a subgraph frequency estimator for motif discovery, which is the focus of this work.

3 Preliminaries

This work studies directed graphs with categorical node and edge attributes. A k-(sub)*graph* G with exactly k nodes[2] is formally represented by a *node type matrix* $X = (x^i)_{1\leq i\leq k} \in \{1, 2, \ldots, a\}^k$ and an *edge type matrix* $E = (e^{ij})_{1\leq i,j\leq k} \in \{1, 2, \ldots, b\}^{k\times k}$, where a, b are the number of types.

The remainder of this section presents the research objective and the theoretical foundation. To facilitate these discussions, several necessary concepts are introduced below. A (sub)graph is *connected* if every pair of nodes is linked by a path. A subgraph S of G is *induced* if every edge of G between nodes in S is also an edge of S. Two graphs are *isomorphic* if there is a one-to-one mapping between their node sets preserving all node and edge attributes. An *induced occurrence* of S in G refers to a specific instance of an induced subgraph within G that is isomorphic to S.

3.1 Research Objective: Frequent Subgraph Discovery

This work aims to discover frequent subgraphs within a target graph set. To formally characterize this objective, it is necessary to define the subgraph counting problem:

Definition 3.1 (Subgraph Counting Problem)**.** Given a (sub)graph S and a graph G, the *subgraph counting problem*, denoted by $\mathrm{SC}(S, G)$, is to determine the number of induced occurrences of S in G. Two occurrences are considered distinct if their node sets differ by at least one node.

For a graph set $\mathcal{G}$, subgraph counting sums occurrences found across all $G \in \mathcal{G}$:

$$\mathrm{SC}(S, \mathcal{G}) = \sum_{G\in\mathcal{G}} \mathrm{SC}(S, G). \tag{1}$$

$\mathcal{D}_k$ denotes all connected induced k-subgraphs in $\mathcal{G}$, partitioned into multiple sets, such that instances are isomorphic within the same set and non-isomorphic across different sets. Each set is represented

[2]For clarity, calligraphic typeface ($\mathcal{G}$) denotes sets; bold typeface ($\mathbf{G}$) denotes random variables, vectors or matrices; and non-bold typeface (G) denotes specific values or instances.

by one subgraph pattern shared by its instances, and the collection of these representatives is denoted by $\mathcal{S}_k = \{S_k^1, S_k^2, \dots\}$. The *relative frequency* of the i-th subgraph pattern S_k^i is defined as:

$$\mathrm{RF}_k(S_k^i, \mathcal{G}) = \mathrm{SC}(S_k^i, \mathcal{G}) \Big(\sum_{j=1}^{|\mathcal{S}_k|} \mathrm{SC}(S_k^j, \mathcal{G}) \Big)^{-1}. \tag{2}$$

The research objective is thus defined as:

Definition 3.2 (Frequent Subgraph Discovery)**.** Given a graph set $\mathcal{G}$ and a subgraph size k, the goal is to efficiently find subgraphs $S \in \mathcal{S}_k$ with high RF_k.

3.2 Theoretical Foundation: Graph Diffusion Models

Graph diffusion models transform samples from a simple prior $\mathbf{G}_T \sim p_T$ (e.g., Gaussian, uniform, or marginal [36]) into the target $\mathbf{G}_0 \sim p_0$ via a learned reverse process. For any instance G_0, its generative probability can be estimated as the expectation over a latent noising trajectory $\{\mathbf{G}_t\}_{t \in \mathcal{T}}$:

$$p_0(G_0) = \mathbb{E}_{\substack{\mathbf{G}_{\Delta t}, \dots, \mathbf{G}_T \\ \sim p_{\Delta t, \dots, T|0}}} \left[p_T(\mathbf{G}_T) \prod_{t \in \mathcal{T}} \frac{p_{t-\Delta t|t}(\mathbf{G}_{t-\Delta t}|\mathbf{G}_t)}{p_{t|t-\Delta t}(\mathbf{G}_t|\mathbf{G}_{t-\Delta t})} \right], \tag{3}$$

where $\mathcal{T} = \{\Delta t, \dots, T\}$ is the time steps. This equation provides a formal way to estimate $p_0(G_0)$.

4 Method

Frequent subgraph discovery in $\mathcal{G}$ relies on evaluating $\mathrm{RF}_k(G_0, \mathcal{G})$ for candidates G_0 in the collection of connected k-subgraphs $\mathcal{D}_k$, [3] which is computationally NP-hard. GraDE circumvents this by constructing a subgraph frequency estimator based on graph diffusion models.

The remainder of this section develops this idea into a complete framework. Section 4.1 establishes the link between subgraph frequency and generative probability. Section 4.2 details the training set preparation by subgraph sampling. Section 4.3 instantiates the graph diffusion estimator. Section 4.4 introduces a beam search framework leveraging this estimator for frequent subgraph discovery.

4.1 From Subgraph Frequency to Generative Probability

GraDE estimates the intractable $\mathrm{RF}_k(G_0, \mathcal{G})$ by aligning the generative distribution of a diffusion model with the true subgraph frequency. Specifically, a graph diffusion model is trained on a representative subset of $\mathcal{D}_k$, where the generative probability $p_0^\theta(G_0)$ serves as a direct surrogate for $\mathrm{RF}_k(G_0, \mathcal{G})$. To justify this surrogate's reliability, the estimation error is decomposed as:

Proposition 4.1 (Error Decomposition of Frequency Estimation)**.** *Let $\pi^*(G_0) = RF_k(G_0, \mathcal{G})$ be the ground-truth subgraph distribution in $\mathcal{D}_k$. Let $\hat{\pi}_0$ and π_0 denote the empirical distribution of M instances sampled from $\mathcal{D}_k$ and its asymptotic limit as $M \to \infty$, respectively. For a learned generative distribution p_0^θ, the estimation error is bounded as follows:*

$$|p_0^\theta(G_0) - \pi^*(G_0)| \le \underbrace{|p_0^\theta(G_0) - \hat{\pi}_0(G_0)|}_{\epsilon_{appx}} + \underbrace{|\hat{\pi}_0(G_0) - \pi_0(G_0)|}_{\epsilon_{rnd}} + \underbrace{|\pi_0(G_0) - \pi^*(G_0)|}_{\epsilon_{sys}}. \tag{4}$$

This decomposition attributes the error to three sources: (i) **approximation error** ϵ_{appx}, resulting from the model's imperfect fit to training data; (ii) **random error** ϵ_{rnd}, arising from statistical fluctuations in finite sampling; (iii) **systematic error** ϵ_{sys}, inherent in the bias of subgraph sampling algorithm.

To tighten the estimation error bound, each component must be minimized. This section focuses on ϵ_{appx} to isolate the intrinsic link between graph diffusion and subgraph frequency from sampling errors (ϵ_{rnd} and ϵ_{sys}, discussed in Section 4.2). Specifically, ϵ_{appx} is bounded by Pinsker's inequality:

$$|p_0^\theta(G_0) - \hat{\pi}_0(G_0)| \le \mathrm{TV}(p_0^\theta, \hat{\pi}_0) \le \sqrt{\frac{1}{2}\mathrm{KL}(\hat{\pi}_0 || p_0^\theta)}, \tag{5}$$

[3] In this section, G_0 denotes a k-subgraph instance from $\mathcal{D}_k$, not a full graph from $\mathcal{G}$.

where $\mathrm{TV}(p_0^\theta, \hat{\pi}_0)$ is the total variation distance, and $\mathrm{KL}(\hat{\pi}_0 \| p_0^\theta)$ is the KL divergence.

Expanding the KL term formally links the approximation error to Maximum Likelihood Estimation:

$$\mathrm{KL}(\hat{\pi}_0 || p_0^\theta) = \mathbb{E}_{G_0 \sim \hat{\pi}_0}[-\log p_0^\theta(G_0)] + \mathbb{E}_{G_0 \sim \hat{\pi}_0}[\log \hat{\pi}_0(G_0)] = \mathrm{NLL}(\theta) + C, \tag{6}$$

where NLL is the negative log-likelihood and C is a constant independent of the model parameters θ. The derivation above establishes that the absolute estimation error is fundamentally bounded by the model's NLL. Importantly, this bound relies solely on the model's capacity to represent discrete distributions rather than any specific diffusion mechanism. This leads to a generalized conclusion:

Proposition 4.2 (Generative Probability as a Surrogate for Subgraph Frequency)**.** *Any generative model trained by maximum likelihood can theoretically approximate the subgraph frequency distribution. Provided p_0^θ has sufficient expressive capacity to fit $\hat{\pi}_0$, the generative probability $p_0^\theta(G_0)$ serves as a direct surrogate for $RF_k(G_0, \mathcal{G})$.*

Graph diffusion models are theoretically suitable because their training objectives are equivalent to maximum-likelihood estimation. Moreover, graph diffusion models have demonstrated strong modeling capability on graph generation tasks, motivating their use as subgraph frequency estimators.

4.2 Subgraph Sampling

Training the graph diffusion estimator requires the dataset $\mathcal{D}_k$. As the subgraph size k increases, the exponential growth makes it impossible to obtain the complete $\mathcal{D}_k$. Fortunately, the estimator can fit the global distribution using only a representative subset, thereby bypassing exhaustive enumeration.

GraDE employs subgraph sampling algorithms to construct the representative subset of $\mathcal{D}_k$. Rather than designing a new sampling algorithm, GraDE treats the sampling module as a pluggable component, enabling the seamless integration of various established uniform subgraph sampling algorithms, such as Rand-ESU [39], ARS, NRS [19], and Rand-FaSE [25].

To eliminate the systematic error ϵ_{sys}, theoretically unbiased algorithms such as Rand-ESU, ARS, or Rand-FaSE can be prioritized. Meanwhile, the random error ϵ_{rnd} diminishes as the sample size M increases, following the statistical convergence guaranteed by the Law of Large Numbers. In terms of efficiency, lightweight algorithms like NRS are ideal for quick representative subset sampling. Details of these algorithms are provided in Appendix F.

4.3 Graph Diffusion Estimator

GraDE is a discovery framework whose estimator can be instantiated using various graph diffusion models. In this work, the estimator is concretely instantiated into three distinct versions based on representative discrete diffusion models: DisCo [41], DiGress [36], and DeFoG [26]. To maintain conciseness, the following formulation focuses on the estimator based on DisCo (denoted as DisCo-E), while the detailed implementations for alternative versions are provided in Appendix D and E.

Forward Transition. Following DisCo [41], the noising process is a continuous-time Markov chain (CTMC) where node and edge attributes are corrupted independently via rate matrices $\mathbf{R}_t^x = \beta(t)\mathbf{R}_x$ and $\mathbf{R}_t^e = \beta(t)\mathbf{R}_e$. Here, $\beta(t) = \alpha\gamma^t \log \gamma$ is the noise schedule, while $\mathbf{R}_x = \mathbf{1}\mathbf{1}^\top - a\mathbf{I}$ and $\mathbf{R}_e = \mathbf{1}\mathbf{1}^\top - b\mathbf{I}$ are constant matrices. For any $0 \le s < t \le T$, this allows the forward transition probability to factorize into exact closed-form solutions[4]:

$$p_{t|s}(G_t|G_s) = \prod_{i=1}^{k} \Big(\exp \int_s^t \beta(u)\mathbf{R}_x du \Big)_{x_s^i x_t^i} \prod_{i,j=1}^{k} \Big(\exp \int_s^t \beta(u)\mathbf{R}_e du \Big)_{e_s^{ij} e_t^{ij}}. \tag{7}$$

Reverse Transition. To instantiate $p_{s|t}$ ($s = t - \Delta t$) required by Eq. (3), this paper derives an analytical form by characterizing the reverse evolution as a Poisson process. While the reverse rates $\hat{\mathbf{R}}_t$ (instantiated via neural posterior $p_{0|t}^\theta$) follow the framework of DisCo, this paper further

[4]For any matrix $\mathbf{M}$, the notation $(\mathbf{M})_{ij}$ denotes the entry at the i-th row and j-th column, where indices i and j correspond to the states in the discrete space Ω.

establishes the reverse transition probability over a finite interval Δt as:

$$p_{s|t}(G_s|G_t) = \prod_{i=1}^{k} p_{s|t}(x_s^i|G_t) \prod_{i,j=1}^{k} p_{s|t}(e_s^{ij}|G_t), \tag{8}$$

where $p_{s|t}(x_s^i|G_t)$ is specified as follows:

$$p_{s|t}(x_s^i|G_t) = \Big(\exp\Big(\Delta t \hat{\mathbf{R}}_t^i(x_t^i, x_t^i)\Big) - 1 + \delta(x_t^i, x_s^i) \Big) \frac{\hat{\mathbf{R}}_t^i(x_t^i, x_s^i)}{\hat{\mathbf{R}}_t^i(x_t^i, x_t^i)}. \tag{9}$$

Here, δ denotes the Kronecker delta. The term $\hat{\mathbf{R}}_t^i(a, b)$ are the instantaneous transition rates from state a to b for nodes. $p_{s|t}(e_s^{ij}|G_t)$ following an identical form. Details are provided in Appendix C.

Algorithm 1 Monte Carlo Estimation

Input: Target subgraph G_0, forward and reverse rate matrices $\{\mathbf{R}_t, \hat{\mathbf{R}}_t\}$, the number of Monte Carlo trials MC, time step Δt, max time T, prior noising distribution p_T
Output: Estimated log-probability $\log \hat{p}_0^\theta(G_0)$
for $i = 1 \to MC$ **do**
  $L_i \leftarrow 0, t \leftarrow \Delta t$ ▷ Initialize log-probability for current path
  while $t \leq T$ **do**
    Sample $G_t \sim p_{t|t-\Delta t}(\cdot|G_{t-\Delta t})$ ▷ Add random noise to $G_{t-\Delta t}$ via Eq. (7)
    $L_i \leftarrow L_i + \log \frac{p_{t-\Delta t|t}(G_{t-\Delta t}|G_t)}{p_{t|t-\Delta t}(G_t|G_{t-\Delta t})}$ ▷ Accumulate log-ratio via Eqs. (7), (8)
    $t \leftarrow t + \Delta t$
  end while
  $L_i \leftarrow L_i + \log p_T(G_T)$ ▷ Path joint probability
end for
return $\log \hat{p}_0^\theta(G_0) = \frac{1}{MC} \sum_{i=1}^{MC} L_i$

Monte Carlo Estimation. Given the transition probabilities derived above, $p_0(G_0)$ in (3) can be estimated via Monte Carlo trials. Each Monte Carlo trial stochastically samples a noising path $\{\mathbf{G}_t\}_{t\in\mathcal{T}}$ from G_0, calculates its joint probability using Eq. (7) and (8). By averaging these path-wise joint probabilities over multiple trials, an estimate of $\hat{p}_0^\theta(G_0)$ is obtained. Notably, this estimation scheme is model-agnostic and applicable to any diffusion framework provided that the forward and reverse transition probabilities are available. For enhanced numerical stability, the estimation is performed in the log-domain as detailed in Alg. 1, with further details provided in Appendix A.

4.4 Beam Search

Algorithm 2 Beam Search

Input: Graph set $\mathcal{G}$, target size k_{max}, top-N threshold N, graph diffusion estimator p^θ.
Output: Top-N frequent subgraphs $\mathcal{F}_{k_{max}}$.
$\mathcal{C}_3 \leftarrow$ Enumerate all 3-subgraphs in $\mathcal{G}$
$\mathcal{F}_3 \leftarrow$ select top-N from $\mathcal{C}_3$
for $k = 3 \to k_{max} - 1$ **do**
  $\mathcal{C}_{k+1} \leftarrow$ ExpandOneNode($\mathcal{G}$, $\mathcal{F}_k$)
  Compute scores $s_g = \log p^\theta(g)$ for all $g \in \mathcal{C}_{k+1}$
  $\mathcal{F}_{k+1} \leftarrow$ select top-N from $\mathcal{C}_{k+1}$
end for
return $\mathcal{F}_{k_{max}}$

Using the graph diffusion estimator, GraDE adopts a beam search framework to discover frequent subgraphs. The process follows a node-incremental expansion strategy. Initially, all connected subgraphs of size $k = 3$ are enumerated, with their exact frequencies and instance locations recorded. In each iteration, the top-N frequent k-subgraphs are selected as seeds. These seeds, with their locations, are used to generate candidate $(k + 1)$-subgraphs by adding an adjacent node, which are then scored by the estimator. Only candidates with high predicted probabilities are retained for the next iteration. The complete procedure and further details are provided in Alg. 2 and Appendix B.

5 Experiments

Section 5.1 details the setup. Section 5.2 compares the GraDE estimator with baselines. Section 5.3 investigates how key hyperparameters influence the performance of the GraDE estimator. Section 5.4 evaluates the overall effectiveness of the GraDE framework in finding frequent subgraph patterns.

5.1 Experimental Setup

Datasets. Three neural architecture datasets are used: synthesized NAS-Bench-101 [43], NAS-Bench-201 [8] and real-world Younger [42] (several orders of magnitude larger than NAS-Bench). To evaluate GraDE estimator's generalization on non-neural domains, HighSchool [20] (social network) and PROTEINS [3] (biological network) are incorporated. See more details in Appendix G.1. Additional results on NAS-Bench-301 [47] and NAS-Bench-NLP [15] are in Appendix H.4.

Key Hyperarameters. The estimator performance depends on the training set, which is determined by three parameters: *subgraph size* k, *sampling method*, and *sampling density* [5]. Another parameter is *estimation rounds*, which is the number of Monte Carlo trials for estimating $p_0(G_0)$.

Evaluation Protocol. The evaluation consists of two parts. First, estimator performance is evaluated by Spearman's rank correlation (Spearman's ρ) and Kendall's rank correlation (Kendall's τ) between predicted scores and ground-truth frequencies obtained from exhaustive k-subgraph enumeration, assessing ranking quality rather than accurate frequency estimation. Second, the overall framework is evaluated for larger k where enumeration is infeasible. This part compares median frequencies of the discovered top-N subgraphs, verified by the VF2 [7] algorithm with a cutoff time. Details are in G.2.

5.2 The Performance of Graph Diffusion Estimator

Table 1: Experimental results on neural architecture datasets. Results (mean±std) are reported for $k \in \{4, 5\}$ at 0.1 sampling density. Evaluation metrics are Spearman's ρ and Kendall's τ. **Bold** and underlined denote the best and second-best results. DisCo-E, DeFoG-E, and DiGress-E are GraDE estimators, using Rand-ESU for training set construction. Estimation rounds are 20 for DisCo-E/DiGress-E/GraphVAE, and 60 for DeFoG-E. "X" indicates timeout for ARS on Younger. Due to exact enumeration costs, Younger ($k = 5$) results are evaluated on a random 10% subset.

Model	Class	NAS-Bench-101		NAS-Bench-201		Younger	
		$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$
Subgraph size $k = 4$							
ARS	Sampling	0.33± 0.008	0.27± 0.007	0.41± 0.005	0.38± 0.004	X	X
NRS	Sampling	0.34± 0.005	0.28± 0.004	0.39± 0.009	0.36± 0.009	0.45± 0.000	0.38± 0.000
Rand-ESU	Sampling	0.36± 0.022	0.29± 0.018	0.39± 0.012	0.36± 0.011	0.46± 0.001	0.38± 0.001
Rand-FaSE	Sampling	0.34± 0.011	0.28± 0.009	0.40± 0.014	0.37± 0.013	0.35± 0.001	0.29± 0.001
GraphVAE	VAE	0.40± 0.021	0.27± 0.017	0.34± 0.058	0.26± 0.048	0.38± 0.002	0.26± 0.002
GNNRanker	GNN	0.60± 0.124	0.44± 0.100	0.61± 0.220	0.48± 0.177	0.32± 0.020	0.22± 0.014
DiGress-E	Diffusion	0.60± 0.077	0.43± 0.063	0.72± 0.065	0.57± 0.042	0.38± 0.008	0.27± 0.006
DeFoG-E	Flow	0.84± 0.010	0.65± 0.006	0.75± 0.060	0.60± 0.038	0.41± 0.013	0.29± 0.009
DisCo-E	Diffusion	**0.88**± 0.023	**0.71**± 0.032	**0.79**± 0.030	**0.66**± 0.029	**0.55**± 0.007	**0.40**± 0.006
Subgraph size $k = 5$							
ARS	Sampling	0.43± 0.001	0.37± 0.001	0.31± 0.004	0.28± 0.003	X	X
NRS	Sampling	0.42± 0.001	0.36± 0.001	0.32± 0.006	0.29± 0.006	0.41± 0.000	0.35± 0.000
Rand-ESU	Sampling	0.44± 0.002	0.38± 0.002	0.33± 0.011	0.30± 0.010	0.42± 0.001	0.35± 0.000
Rand-FaSE	Sampling	0.42± 0.002	0.36± 0.002	0.30± 0.007	0.27± 0.007	0.33± 0.007	0.28± 0.006
GraphVAE	VAE	0.42± 0.011	0.30± 0.009	0.70± 0.009	0.56± 0.008	0.32± 0.003	0.23± 0.002
GNNRanker	GNN	0.87± 0.025	0.71± 0.029	0.83± 0.033	0.68± 0.035	0.32± 0.015	0.23± 0.011
DiGress-E	Diffusion	0.88± 0.003	0.73± 0.004	0.85± 0.001	0.70± 0.001	0.40± 0.008	0.28± 0.006
DeFoG-E	Flow	0.91± 0.007	0.75± 0.009	0.61± 0.004	0.47± 0.003	0.47± 0.002	0.33± 0.002
DisCo-E	Diffusion	**0.94**± 0.002	**0.81**± 0.002	**0.87**± 0.016	**0.73**± 0.020	**0.56**± 0.002	**0.40**± 0.001

[5]The ratio of non-isomorphic k-subgraph types in the training set to that in the dataset. For a dataset with 5 types, 0.4 means the training set contains instances from 2 types (e.g., $\{A, A, A, B\}$).

To our knowledge, this work is the first to apply graph diffusion models for subgraph frequency estimation. Besides DisCo-E in Section 4.3, this section further reproposes several representative graph generative models, including DiGress [36], DeFoG [26], and GraphVAE [33], as subgraph frequency estimators for comparison. To compare with direct GNN-based frequency prediction, GNNRanker is additionally implemented as a supervised frequency estimator based on GIN with edge features [11]. These models are evaluated alongside sampling-based methods, including ARS, NRS [19], Rand-ESU [39], and Rand-FaSE [25]. For a fair comparison, the sampling density for all methods is fixed to 0.1. For generative methods and GNNRanker, Rand-ESU serves as the sampling method for constructing the training set. The performance of all methods is evaluated across subgraph sizes $k \in \{4, 5\}$, as obtaining ground truth for larger k on Younger is computationally prohibitive.

As shown in Table 1, DisCo-E achieves the best performance across all neural architecture datasets. Specifically, DisCo-E significantly improves Spearman's ρ by up to 114% on synthesized NAS-Bench-101 and 33% on real-world Younger (both vs. Rand-ESU, $k = 5$). Furthermore, diffusion-based and flow-based methods, including DiGress-E and DeFoG-E, significantly outperform sampling and VAE-based methods on synthesized datasets, reflecting their superior capability in modeling underlying frequency distributions. These results demonstrate the superiority of using graph diffusion models as subgraph frequency estimators, with DisCo-E consistently achieving the best results.

Table 2: Spearman's ρ beyond neural architecture datasets. **Bold** and underlined denote the best and second-best performance. ARS is omitted due to prohibitive costs on larger graphs.

Dataset / k	NRS	Rand-ESU	Rand-FaSE	GraphVAE	DiGress-E	DeFoG-E	DisCo-E
HighSchool / 4	0.29	0.30	0.33	0.37	0.41	<u>0.42</u>	**0.45**
HighSchool / 5	0.29	0.32	0.31	0.35	0.37	**0.42**	**0.42**
PROTEINS / 4	0.46	0.44	0.43	0.39	<u>0.48</u>	<u>0.48</u>	**0.54**
PROTEINS / 5	0.45	0.46	0.46	0.36	<u>0.51</u>	0.48	**0.60**

To evaluate GraDE's robustness beyond neural architectures, experiments are conducted on social (HighSchool) and biological (PROTEINS) networks. In Table 2, DisCo-E consistently achieves the best performance across all configurations in Spearman's ρ, with up to 30% improvement (vs. Rand-ESU, PROTEINS, $k = 5$). All GraDE estimators (DisCo-E, DiGress-E, DeFoG-E) significantly outperform sampling-based baselines, confirming GraDE's generalization on non-neural domains.

5.3 Ablation Study: Impact of Key Hyperarameters on Estimator's Performance

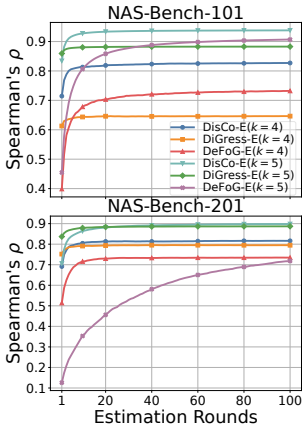


Figure 2: Spearman's ρ vs. estimation rounds. GraDE estimators are paired with the Rand-ESU at 0.1 sampling density. Legend parameter k denotes the subgraph size.

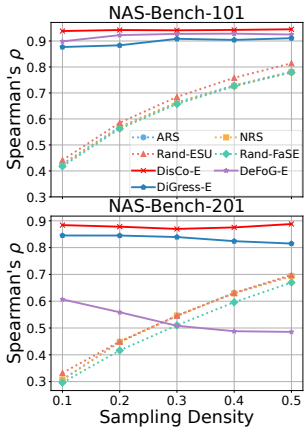


Figure 3: Spearman's ρ vs. sampling densities. All estimators use Rand-ESU with $k = 5$. Estimation rounds are 20 for DisCo-E/DiGress-E and 60 for DeFoG-E.

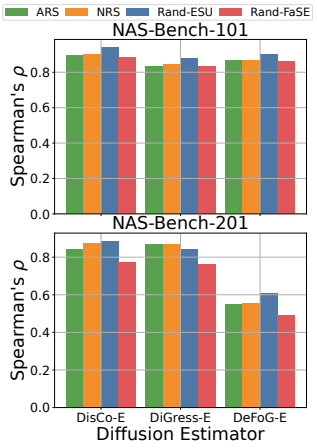


Figure 4: Spearman's ρ across sampling methods. Results are at $k = 5$ and 0.1 sampling density. Estimation rounds are 20 for DisCo-E/DiGress-E and 60 for DeFoG-E.

This section investigates the impact of estimation rounds, sampling density, and sampling method.

Impact of Estimation Rounds. The estimation rounds influence both performance and computational cost. Figure 2 presents Spearman's ρ across different methods and subgraph sizes ($k \in \{4, 5\}$) as estimation rounds increase. For all estimators, Spearman's ρ initially rises and then stabilizes. DisCo-E and DiGress-E converge at approximately 20 rounds, while DeFoG-E stabilizes around 60 rounds. These elbow points represent an optimal trade-off between performance and efficiency, indicating that GraDE's Monte Carlo process converges rapidly with limited iterations.

Impact of Sampling Density. The sampling density is directly related to the total number of sampled subgraphs. Since the sampling cost grows exponentially with the sampling density (see Appendix H.3), a high sampling density leads to prohibitive computational costs. Figure 3 shows the performance of various methods as the sampling density increases from 0.1 to 0.5. For sampling-based methods, their performance improves with higher sampling densities. In contrast, the performance of graph diffusion estimators is stable, approaching their optimum at a sampling density of only 0.1. Notably, DisCo-E achieves higher performance at 0.1 density than all sampling-based methods at 0.5. These results demonstrate that GraDE can achieve superior performance even under sparse sampling conditions, which is crucial for mitigating the exponentially increasing costs of subgraph sampling.

Impact of Sampling Method. Figure 4 compares the performance of various graph diffusion estimators when paired with different sampling methods. Results on NAS-Bench-101 and NAS-Bench-201 show that the performance for all estimators remains stable across different sampling methods, verifying the robustness of GraDE. Nevertheless, Rand-ESU consistently achieves the best or near-best results in most cases, which is likely due to its unbiased sampling nature.

5.4 Discovery Capability of the GraDE Framework

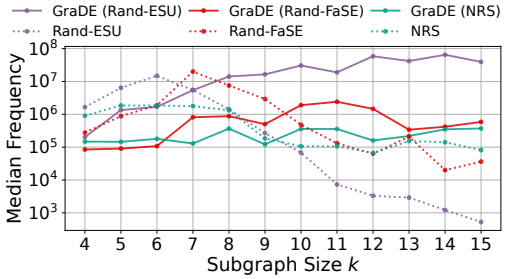


Figure 5: Median frequencies of top-50 discovered subgraphs by different methods.

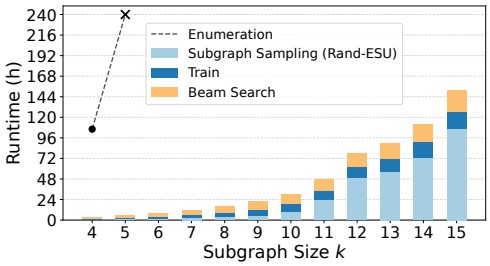


Figure 6: Runtime combination of GraDE (Rand-ESU) vs. Enumeration.

To evaluate GraDE's capability to find frequent subgraph patterns, experiments are conducted on the real-world Younger dataset for $k \in \{4, \ldots, 15\}$. The GraDE framework, specifically using DisCo-E trained on samples from NRS, Rand-ESU, or Rand-FaSE, is compared against these sampling-based methods. For each k, the sampling baselines draw 40,000 instances, while GraDE utilizes the same samples for training before initiating the search. Following the beam search strategy in Section 4.4, GraDE starts from 3-subgraphs and retains the top-1000 candidates per iteration via DisCo-E scores. Finally, the true frequencies of the top-50 discovered subgraphs are verified via VF2 algorithm.

Figure 5 presents the median frequencies of the top-50 discovered subgraphs as quantitative measures of discovery quality. As shown, a performance crossover occurs as k increases. For small k, the subgraph search space remains relatively limited, allowing sampling-based methods to provide reliable frequency signals. As k grows, the search space expands exponentially and sampling becomes increasingly sparse, whereas GraDE can generalize from well-sampled candidates to candidates with limited sampling coverage, leading to a substantial advantage for $k \geq 10$. In this range, GraDE (Rand-ESU) outperforms Rand-ESU by over $100\times$, while GraDE (Rand-FaSE) and GraDE (NRS) achieve $10\times$ and $2.5\times$ average improvements over their respective sampling baselines. Notably, GraDE (Rand-ESU) emerges as the best overall combination starting from $k = 8$. This advantage over the best sampling-based method exceeds $30\times$ in large-scale scenarios ($k \geq 10$), demonstrating GraDE's superior capability in discovering large-scale frequent subgraph patterns.

Figure 6 decomposes the cumulative runtime of GraDE (Rand-ESU) from $k = 4$ into sampling, training, and beam search. As k increases, sampling occupies an expanding proportion of the total time, whereas the additional overhead from model training and beam search becomes relatively marginal. Notably, while the sampling time grows rapidly with k, the training and search costs exhibit modest growth. This efficiency contrasts with exact enumeration, which becomes computationally prohibitive as k increases. These results demonstrate the computational feasibility of GraDE.

5.5 Impact of Estimator and Search Strategy on the GraDE Framework

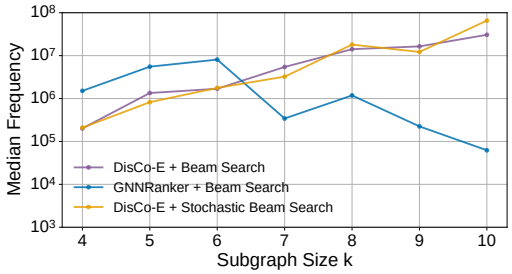


Figure 7: Median frequencies of top-50 subgraphs under different estimators and search strategies.

This section investigates how the estimator and search strategy affect the discovery capability of the GraDE framework. Experiments are conducted on Younger for $k \in \{4, \ldots, 10\}$, following the Rand-ESU setting in Section5.4. Using DisCo-E with beam search as the reference, the estimator is replaced with GNNRanker, or the search strategy is replaced with stochastic beam search.

As shown in Figure 7, DisCo-E achieves comparable discovery performance with standard and stochastic beam search despite the randomness introduced by stochastic selection, suggesting the robustness of the GraDE framework to this change in search strategy. For the estimator comparison, GNNRanker discovers higher-frequency patterns for smaller k, whereas DisCo-E achieves substantially higher median frequencies for $k \geq 7$, demonstrating the advantage of the GraDE estimator in discovering large-scale frequent subgraph patterns.

6 Conclusion

This paper proposes GraDE, a novel framework for frequent subgraph discovery. By leveraging graph diffusion models to score subgraph typicality, GraDE effectively overcomes the exponential complexity of enumeration-based methods and the limited scalability of sampling-based methods. Comprehensive experiments validate the effectiveness of the GraDE estimator and the overall GraDE framework. While GraDE reduces runtime to a feasible level, there still remains room for further efficiency optimization during training and search. Future work will focus on efficiency optimization and extending GraDE to more diverse graph domains.

Limitations

GraDE's discovery performance can fall below sampling-based methods for small k, potentially due to estimator errors and the early pruning of promising subgraph patterns during beam search. In addition, its effectiveness depends on the diffusion model's ability to capture the underlying subgraph distribution, and inaccurate distribution modeling may reduce estimation quality and subsequent discovery performance.

Acknowledgments

Supported by the Innovation Funding of ICT, CAS under Grant No. E661110000.

References

[1] Rohan Asthana, Joschua Conrad, Youssef Dawoud, Maurits Ortmanns, and Vasileios Belagiannis. Multi-conditioned graph diffusion for neural architecture search. *Transactions on Machine Learning Research*, 2024.

[2] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating graphs via random walks. In *International Conference on Machine Learning (ICML)*, 2018.

[3] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, January 2005.

[4] Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. In *International Conference on Machine Learning (ICML)*, 2024.

[5] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *ArXiv*, abs/1805.11973, 2018.

[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2018.

[7] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 26(10):1367–1372, 2004.

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

[9] Wenqi Fan, Cheng-Yun Karen Liu, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. Generative diffusion models on graphs: Methods and applications. In *International Joint Conference on Artificial Intelligence*, 2023.

[10] Tomaž Hočevar and Janez Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565, 2014.

[11] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[12] Johan Ludwig William Valdemar Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30(1):175–193, 1906.

[13] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning (ICML)*, 2022.

[14] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, July 2004.

[15] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. NAS-Bench-NLP: Neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022.

[16] Jianxin Li, Qingyun Sun, Hao Peng, Beining Yang, Jia Wu, and Philip S. Yu. Adaptive subgraph neural network with reinforced critical structure mining. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(7):8063–8080, 2023.

[17] Yanxi Li, Minjing Dong, Yunhe Wang, and Chang Xu. Neural architecture search in a proxy validation loss landscape. In *International Conference on Machine Learning (ICML)*, 2020.

[18] Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[19] Xuesong Lu and Stéphane Bressan. Sampling Connected Induced Subgraphs Uniformly at Random. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, 2012.

[20] Rossana Mastrandrea, Julie Fournet, and Alain Barrat. Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PLOS ONE*, 10(9):1–26, 09 2015.

[21] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[22] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[23] Carlos Oliver, Dexiong Chen, Vincent Mallet, Pericles Philippopoulos, and Karsten Borgwardt. Approximate network motif mining via graph learning, 2022. arXiv:2206.01008.

[24] Pedro Paredes and Pedro Ribeiro. Towards a faster network-centric subgraph census. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2013.

[25] Pedro Paredes and Pedro Ribeiro. Rand-FaSE: Fast approximate subgraph census. *Social Network Analysis and Mining*, 5(1):17, 2015.

[26] Yiming Qin, Manuel Madeira, Dorina Thanou, and Pascal Frossard. DeFoG: Discrete flow matching for graph generation. In *International Conference on Machine Learning (ICML)*, 2025.

[27] Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys*, 54(2):1–36, 2022.

[28] Pedro Ribeiro and Fernando Silva. Efficient Subgraph Frequency Estimation with G-Tries. In *Algorithms in Bioinformatics*, pages 238–249, 2010.

[29] Pedro Ribeiro and Fernando Silva. G-tries: An efficient data structure for discovering network motifs. In *ACM Symposium on Applied Computing (SAC)*, 2010.

[30] Tanay Kumar Saha and Mohammad Al Hasan. Finding network motifs using MCMC sampling. In *Complex Networks VI*, pages 13–24, 2015.

[31] Mohammad Salameh, Keith Mills, Negar Hassanpour, Fred Han, Shuting Zhang, Wei Lu, Shangling Jui, CHUNHUA ZHOU, Fengyu Sun, and Di Niu. AutoGO: Automated computation graph optimization for neural network evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[32] Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. Learning gradient fields for molecular conformation generation. In *International Conference on Machine Learning (ICML)*, 2021.

[33] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks (ICANN)*, 2018.

[34] Antoine Siraudin, Fragkiskos D. Malliaros, and Christopher Morris. Cometh: A continuous-time discrete-state graph diffusion model. *Transactions on Machine Learning Research*, 2025.

[35] Pedro C. Vieira, Miguel E. P. Silva, and Pedro Manuel Pinto Ribeiro. Studying and improving graph neural network-based motif estimation, 2025. arXiv:2506.15709.

[36] 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 (ICLR)*, 2023.

[37] Xingchen Wan, Binxin Ru, Pedro M Esperança, and Zhenguo Li. On redundancy and diversity in cell-based neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2022.

[38] Jianxin Wang, Yuannan Huang, Fang-Xiang Wu, and Yi Pan. Symmetry compression method for discovering network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(6):1776–1789, 2012.

[39] Sebastian Wernicke. Efficient Detection of Network Motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359, 2006.

[40] Ruofan Wu, Zhen Zheng, Feng Zhang, Chuanjie Liu, Zaifeng Pan, Jidong Zhai, and Xiaoyong Du. PluS: Highly efficient and expandable ml compiler with pluggable graph schedules. In *USENIX Annual Technical Conference (USENIX ATC)*, 2025.

[41] Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. Discrete-state Continuous-time Diffusion for Graph Generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[42] Zhengxin Yang, Wanling Gao, Luzhou Peng, Yunyou Huang, Fei Tang, and Jianfeng Zhan. Younger: The first dataset for artificial intelligence-generated neural network architecture. *ArXiv*, abs/2406.15132, 2024.

[43] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin P. Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning (ICML)*, 2019.

[44] Rex Ying, Tianyu Fu, Andrew Wang, Jiaxuan You, Yu Wang, and Jure Leskovec. Representation learning for frequent subgraph mining, 2024. arXiv:2402.14367.

[45] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning (ICML)*, 2018.

[46] Zhaoning Yu and Hongyang Gao. MAGE: Model-level graph neural networks explanations via motif-based graph generation. In *International Conference on Learning Representations (ICLR)*, 2025.

[47] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks. In *International Conference on Learning Representations (ICLR)*, 2022.

[48] Shichang Zhang, Ziniu Hu, Arjun Subramonian, and Yizhou Sun. Motif-driven contrastive learning of graph representations. *IEEE Transactions on Knowledge and Data Engineering*, 36(8):4063–4075, 2024.

Appendix Overview

Appendix provides additional details organized as follows:

1. Appendix A: Monte Carlo Estimation.
2. Appendix B: Details of the Beam Search.
3. Appendix C: Proofs of DisCo Estimator (DisCo-E).
4. Appendix D: DiGress Estimator (DiGress-E).
5. Appendix E: DeFoG Estimator (DeFoG-E).
6. Appendix F: Subgraph Sampling Methods.
7. Appendix G: Further Implementation Details.
8. Appendix H: Additional Experimental Results.
9. Appendix I: Similar but Distinct Work.

A Monte Carlo Estimation

By applying Jensen's inequality [12], $\log p_0(G_0)$ is bounded as follows:

$$\log p_0(G_0) \geq \mathbb{E}_{\substack{\mathbf{G}_{\Delta t},\ldots,\mathbf{G}_T \\ \sim p_{\Delta t,\ldots,T|0}}} \left[\log \Big(p_T(\mathbf{G}_T) \prod_{t\in\mathcal{T}} \frac{p_{t-\Delta t|t}(\mathbf{G}_{t-\Delta t}|\mathbf{G}_t)}{p_{t|t-\Delta t}(\mathbf{G}_t|\mathbf{G}_{t-\Delta t})} \Big) \right] \tag{10}$$

$$= \mathbb{E}_{\substack{\mathbf{G}_{\Delta t},\ldots,\mathbf{G}_T \\ \sim p_{\Delta t,\ldots,T|0}}} \left[\log p_T(\mathbf{G}_T) + \sum_{t\in\mathcal{T}} \Big(\log p_{t-\Delta t|t}(\mathbf{G}_{t-\Delta t}|\mathbf{G}_t) - \log p_{t|t-\Delta t}(\mathbf{G}_t|\mathbf{G}_{t-\Delta t}) \Big) \right], \tag{11}$$

where $\mathcal{T} = \{\Delta t, \ldots, T\}$. The right-hand side is the evidence lower bound (ELBO). This work utilizes this lower bound to estimate $\log p_0(G_0)$ due to its numerical stability.

The relationship between ELBO and $\log p_0(G_0)$ can be characterized through the discrepancy between the forward and learned reverse trajectory distributions. Let $Z = (G_{\Delta t}, \ldots, G_T)$ denote the noising trajectory. Writing the learned model explicitly with superscript θ, these distributions are

$$P(G_0, Z) = p_0(G_0) \prod_{t\in\mathcal{T}} p_{t|t-\Delta t}(G_t|G_{t-\Delta t}), \tag{12}$$

$$P^\theta(G_0, Z) = p_T(G_T) \prod_{t\in\mathcal{T}} p^\theta_{t-\Delta t|t}(G_{t-\Delta t}|G_t). \tag{13}$$

The ELBO and model log-likelihood can then be expressed as

$$\mathrm{ELBO}(G_0) = \mathbb{E}_{Z\sim P(\cdot|G_0)} \left[\log \frac{P^\theta(G_0, Z)}{P(Z|G_0)} \right], \tag{14}$$

$$\log p^\theta_0(G_0) = \log \mathbb{E}_{Z\sim P(\cdot|G_0)} \left[\frac{P^\theta(G_0, Z)}{P(Z|G_0)} \right]. \tag{15}$$

The latter can be approximated by log-mean-exp over sampled trajectory log-weights, whereas the ELBO averages these log-weights directly.

The candidate-dependent gap is

$$\Delta^\theta(G_0) = \log p^\theta_0(G_0) - \mathrm{ELBO}(G_0) \tag{16}$$

$$= \mathrm{KL}\left(P(Z|G_0) \| P^\theta(Z|G_0)\right). \tag{17}$$

Thus, the gap measures the discrepancy between the forward noising trajectory and the model posterior over trajectories. Applying the KL chain rule gives

$$\mathrm{KL}(P\|P^\theta) = \mathrm{KL}(p_0\|p^\theta_0) + \mathbb{E}_{G_0\sim p_0}[\Delta^\theta(G_0)]. \tag{18}$$

Therefore,

$$\mathbb{E}_{G_0\sim p_0}[\Delta^\theta(G_0)] \le \mathrm{KL}(P\|P^\theta). \tag{19}$$

Assuming that the terminal forward distribution matches the prior p_T, decomposing the trajectory KL along the reverse Markov chain yields

$$\mathrm{KL}(P\|P^\theta) = \sum_{t\in\mathcal{T}} \mathbb{E}_{G_t\sim p_t}\left[\mathrm{KL}\left(p_{t-\Delta t|t}(\cdot|G_t)\|p^\theta_{t-\Delta t|t}(\cdot|G_t)\right)\right]. \tag{20}$$

For reverse transitions obtained by applying the same forward-process-conditioned Markov kernel to $p_{0|t}(\cdot|G_t)$ and $p^\theta_{0|t}(\cdot|G_t)$, the data-processing inequality gives

$$\mathrm{KL}\left(p_{t-\Delta t|t}(\cdot|G_t)\|p^\theta_{t-\Delta t|t}(\cdot|G_t)\right) \le \mathrm{KL}\left(p_{0|t}(\cdot|G_t)\|p^\theta_{0|t}(\cdot|G_t)\right). \tag{21}$$

Combining these results,

$$\mathbb{E}_{G_0\sim p_0}[\Delta^\theta(G_0)] \le \sum_{t\in\mathcal{T}} \mathbb{E}_{G_t\sim p_t}\left[\mathrm{KL}\left(p_{0|t}(\cdot|G_t)\|p^\theta_{0|t}(\cdot|G_t)\right)\right]. \tag{22}$$

The expected ELBO gap is therefore bounded by the denoising posterior error. Under exact posterior matching, this bound vanishes. Since $\Delta^\theta(G_0)$ is nonnegative, the candidate-wise gap also vanishes for every G_0 with $p_0(G_0) > 0$.

Thus, the lower bound serves as a principled proxy for the model log-likelihood. Based on this derivation, Alg. 1 averages trajectory log-weights across Monte Carlo trials to obtain the ranking score.

B Details of the Beam Search

This section describes the node-incremental expansion strategy in GraDE, providing additional technical details to supplement Section 4.4. Alg. 3 provides the complete procedure.

The expansion process begins by using the pre-stored occurrences $\mathcal{I}_g$ associated with each frequent pattern g in $\mathcal{F}_k$. The algorithm performs a localized expansion by identifying nodes v that are adjacent to each specific instance I_g. By adding an adjacent node to each I_g, the algorithm forms a set of $(k+1)$-node candidate instances $\mathcal{R}_{k+1}$ that naturally reside within $\mathcal{G}$.

To manage redundancy, the algorithm deduplicates these candidates by grouping isomorphic instances into unique structural patterns. Specifically, for each instance in $\mathcal{R}_{k+1}$, the algorithm maps it to a unique structural fingerprint via graph hashing. This hash-based approach allows the algorithm to construct a dictionary $\mathcal{C}_{k+1}$, where all isomorphic instances are aggregated under their corresponding hash key. This organization ensures that each unique subgraph pattern is evaluated by the graph diffusion estimator only once, while the associated occurrence lists are preserved to facilitate frequency verification and further expansion in the subsequent iteration.

C Proofs of DisCo Estimator (DisCo-E)

C.1 Forward Probability

The evolution of the forward transition probability $p_{t|s}(y|x)$ for $y, x \in \Omega$ is governed by the Kolmogorov forward equation:

$$\frac{d}{dt}p_{t|s}(y|x) = \sum_{z\in\Omega} p_{t|s}(z|x)\mathbf{R}_t(z,y). \tag{23}$$

Let $\mathbf{P}_{t|s} \in \mathbb{R}^{|\Omega|\times|\Omega|}$ be the forward transition probability matrix where $(\mathbf{P}_{t|s})_{xy} = p_{t|s}(y|x)$. Eq. (23) is then expressed in matrix form as:

$$\frac{d}{dt}\mathbf{P}_{t|s} = \mathbf{P}_{t|s}\mathbf{R}_t, \tag{24}$$

subject to the initial condition $\mathbf{P}_{s|s} = \mathbf{I}$. The proposed solution $\mathbf{P}_{t|s} = \exp\left(\int_s^t \mathbf{R}_u du\right)$ can be verified by substituting it into Eq. (24). Under the commutativity assumption $\mathbf{R}_t\mathbf{R}_s = \mathbf{R}_s\mathbf{R}_t$, the

Algorithm 3 Expand One Node in Beam Search

Input: Graph set $\mathcal{G}$, frequent k-subgraph patterns $\mathcal{F}_k$.
Output: Candidate $(k+1)$-subgraph patterns $\mathcal{C}_{k+1}$.
function ExpandOneNode($\mathcal{G}$, $\mathcal{F}_k$)
    $\mathcal{R}_{k+1} \leftarrow \emptyset$ // Set of raw candidate instances
    for $g \in \mathcal{F}_k$ **do**
        $\mathcal{I}_g \leftarrow$ all occurrences of g in $\mathcal{G}$ // Pre-stored occurrences
        for $I_g \in \mathcal{I}_g$ **do**
            $N(I_g) \leftarrow \{v \in V(\mathcal{G}) \setminus I_g \mid \exists u \in I_g, (u,v) \in E(\mathcal{G}) \vee (v,u) \in E(\mathcal{G})\}$
            for $v \in N(I_g)$ **do**
                $\mathcal{R}_{k+1} \leftarrow \mathcal{R}_{k+1} \cup \{I_g \cup \{v\}\}$
            end for
        end for
    end for
    $\mathcal{C}_{k+1} \leftarrow \{\}$ // Dictionary mapping {GraphHash: Instances}
    for $I'_g \in \mathcal{R}_{k+1}$ **do**
        $h \leftarrow$ GraphHash(I'_g)
        $\mathcal{C}_{k+1}[h] \leftarrow \mathcal{C}_{k+1}[h] \cup \{I'_g\}$
    end for
    return $\mathcal{C}_{k+1}$
end function

integral commutes with its derivative such that $\left(\int_s^t \mathbf{R}_u du\right) \mathbf{R}_t = \mathbf{R}_t \left(\int_s^t \mathbf{R}_u du\right)$. Expanding the matrix exponential as a power series yields:

$$\mathbf{P}_{t|s} = \sum_{i=0}^{+\infty} \frac{1}{i!} \left(\int_s^t \mathbf{R}_u du\right)^i. \tag{25}$$

Substituting this expression into the left-hand side (LHS) of Eq. (24), the derivative is computed as:

$$\begin{aligned}
\text{LHS} &= \frac{d}{dt} \sum_{i=0}^{+\infty} \frac{1}{i!} \left(\int_s^t \mathbf{R}_u du\right)^i && (26)\\
&= \sum_{i=1}^{+\infty} \frac{1}{i!} \cdot i \left(\int_s^t \mathbf{R}_u du\right)^{i-1} \frac{d}{dt} \left(\int_s^t \mathbf{R}_u du\right) && (27)\\
&= \sum_{i=1}^{+\infty} \frac{1}{(i-1)!} \left(\int_s^t \mathbf{R}_u du\right)^{i-1} \mathbf{R}_t && (28)\\
&= \left(\sum_{j=0}^{+\infty} \frac{1}{j!} \left(\int_s^t \mathbf{R}_u du\right)^j\right) \mathbf{R}_t && (29)\\
&= \mathbf{P}_{t|s} \mathbf{R}_t && (30)\\
&= \text{RHS} && (31)
\end{aligned}$$

By taking the entry at row x and column y of the matrix $\mathbf{P}_{t|s}$, the forward transition probability is obtained as:

$$p_{t|s}(y|x) = \left[\exp\left(\int_s^t \mathbf{R}_u du\right)\right]_{xy} \tag{32}$$

C.2 Reverse Probability

The reverse transition rate matrix $\hat{\mathbf{R}}_t$ is defined through the detailed balance condition with respect to the forward process:

$$p_t(y)\hat{\mathbf{R}}_t(y,x) = p_t(x)\mathbf{R}_t(x,y). \tag{33}$$

For any two states $x, y \in \Omega$, the entry at row x and column y of the reverse rate matrix is given by:

$$\hat{\mathbf{R}}_t(y, x) = \mathbf{R}_t(x, y)\frac{p_t(x)}{p_t(y)}. \tag{34}$$

The number of transitions occurring within a small time interval $\Delta t = t - s$ in a continuous-time Markov process follows a Poisson distribution. Let $\mathbf{K}$ be the random variable representing the number of transitions from state y during Δt, with the rate parameter $\lambda = -\Delta t\hat{\mathbf{R}}_t(y, y)$. The probability of observing exactly k transitions is given by:

$$P(\mathbf{K} = k) = \frac{(\lambda)^k e^{-\lambda}}{k!}. \tag{35}$$

For a sufficiently small step Δt, the probability of more than one transition occurring can be ignored, allowing for the approximation $P(\mathbf{K} = 1) \approx 1 - P(\mathbf{K} = 0)$. The reverse transition probability $p_{s|t}(x|y)$ is then derived based on whether a state transition occurs.

If no transition occurs ($x = y$), the probability is simply the zero-event case of the Poisson distribution:

$$p_{s|t}(x|y) = P(\mathbf{K} = 0) = \exp\left(\Delta t\hat{\mathbf{R}}_t(y, y)\right). \tag{36}$$

If a transition occurs ($x \neq y$), the probability is the product of the probability that a transition is initiated and the conditional probability of jumping to state x:

$$p_{s|t}(x|y) = (1 - P(\mathbf{K} = 0)) \cdot \frac{\hat{\mathbf{R}}_t(y, x)}{\sum_{z \neq y} \hat{\mathbf{R}}_t(y, z)} \tag{37}$$

$$= \left(1 - \exp\left(\Delta t\hat{\mathbf{R}}_t(y, y)\right)\right) \frac{\hat{\mathbf{R}}_t(y, x)}{-\hat{\mathbf{R}}_t(y, y)} \tag{38}$$

$$= \left(\exp\left(\Delta t\hat{\mathbf{R}}_t(y, y)\right) - 1\right) \frac{\hat{\mathbf{R}}_t(y, x)}{\hat{\mathbf{R}}_t(y, y)}. \tag{39}$$

Combining the two cases above, the reverse transition probability can be summarized in a unified form:

$$p_{s|t}(x|y) = \left(\exp\left(\Delta t\hat{\mathbf{R}}_t(y, y) - 1 + \delta(y, x)\right)\right) \frac{\hat{\mathbf{R}}_t(y, x)}{\hat{\mathbf{R}}_t(y, y)}. \tag{40}$$

C.3 Factorized Forward Probability

Under the factorization assumption, each node i and edge (i, j) evolves independently according to its respective forward transition rate matrix. The forward rate matrices for a node and an edge are defined as $\mathbf{R}_t^X = \beta(t)\mathbf{R}_x$ and $\mathbf{R}_t^E = \beta(t)\mathbf{R}_e$, where $\Omega_x = \{1, 2, \cdots, a\}$ and $\Omega_e = \{1, 2, \cdots, b\}$ denote their state spaces. The evolution of the transition probabilities $p_{t|s}(x_t^i|x_s^i)$ and $p_{t|s}(e_t^{ij}|e_s^{ij})$ is governed by the following component-wise Kolmogorov forward equations:

$$\frac{d}{dt}p_{t|s}(x_t^i|x_s^i) = \sum_{x \in \Omega_x} p_{t|s}(x|x_s^i)\beta(t)\mathbf{R}_x(x, x_t^i), \tag{41}$$

$$\frac{d}{dt}p_{t|s}(e_t^{ij}|e_s^{ij}) = \sum_{e \in \Omega_e} p_{t|s}(e|e_s^{ij})\beta(t)\mathbf{R}_e(e, e_t^{ij}). \tag{42}$$

Since these equations share the same mathematical structure as Eq. (23) in Appendix C.1, the analytical solutions can be obtained by following an identical derivation from Eq. (23) to Eq. (31). Applying the matrix exponential form to each component yields:

$$p_{t|s}(x_t^i|x_s^i) = \left(\exp(\int_s^t \beta(u)\mathbf{R}_x du)\right)_{x_s^i x_t^i}, \tag{43}$$

$$p_{t|s}(e_t^{ij}|e_s^{ij}) = \left(\exp(\int_s^t \beta(u)\mathbf{R}_e du)\right)_{e_s^{ij} e_t^{ij}}. \tag{44}$$

Due to the independence of nodes and edges in the forward process, the joint transition probability $p_{t|s}(G_t|G_s)$ is the product of these individual transition probabilities.

Based on Eq. (43) and Eq. (44), the final expression of the forward transition probability is:

$$p_{t|s}(G_t|G_s) = \prod_{i=1}^{k} \left(\exp(\int_s^t \beta(u)\mathbf{R}_x du) \right)_{x_s^i x_t^i} \prod_{i,j=1}^{k} \left(\exp(\int_s^t \beta(u)\mathbf{R}_e du) \right)_{e_s^{ij} e_t^{ij}}. \tag{45}$$

C.4 Factorized Reverse Probability

Under the factorization assumption, the reverse transition rate matrix for each node i at time t, denoted as $\hat{\mathbf{R}}_t^i$, is derived from the detailed balance condition:

$$\hat{\mathbf{R}}_t^i(x_t^i, \tilde{x}_t^i) = \mathbf{R}_t^X(\tilde{x}_t^i, x_t^i) \frac{p_t(\tilde{x}_t^i)}{p_t(x_t^i)}. \tag{46}$$

By marginalizing over the initial states $x_0^i \in \Omega_x = \{1, 2, \cdots, a\}$ and applying Bayes' rule, the ratio of marginal probabilities is expanded as follows:

$$\frac{p_t(\tilde{x}_t^i)}{p_t(x_t^i)} = \frac{\sum_{x_0^i \in \Omega_x} p_{t,0}(\tilde{x}_t^i, x_0^i)}{p_t(x_t^i)} \tag{47}$$

$$= \frac{\sum_{x_0^i \in \Omega_x} p_{t|0}(\tilde{x}_t^i|x_0^i) p_0(x_0^i)}{p_t(x_t^i)} \tag{48}$$

$$= \sum_{x_0^i \in \Omega_x} \frac{p_{t|0}(\tilde{x}_t^i|x_0^i) p_0(x_0^i) p_{0|t}(x_0^i|x_t^i)}{p_t(x_t^i) p_{0|t}(x_0^i|x_t^i)} \tag{49}$$

$$= \sum_{x_0^i \in \Omega_x} \frac{p_{t|0}(\tilde{x}_t^i|x_0^i) p_0(x_0^i) p_{0|t}(x_0^i|x_t^i)}{p_0(x_0^i) p_{t|0}(x_t^i|x_0^i)} \tag{50}$$

$$= \sum_{x_0^i \in \Omega_x} p_{0|t}(x_0^i|x_t^i) \frac{p_{t|0}(\tilde{x}_t^i|x_0^i)}{p_{t|0}(x_t^i|x_0^i)}. \tag{51}$$

In practice, since the state of a single node is conditioned on the entire graph structure, the term $p_{0|t}(x_0^i|x_t^i)$ is replaced by the posterior $p_{0|t}(x_0^i|G_t)$, which is approximated by a neural network:

$$\frac{p_t(\tilde{x}_t^i)}{p_t(x_t^i)} = \sum_{x_0^i \in \Omega_x} p_{0|t}(x_0^i|G_t) \frac{p_{t|0}(\tilde{x}_t^i|x_0^i)}{p_{t|0}(x_t^i|x_0^i)}. \tag{52}$$

Substituting Eq. 43 into the rate matrix expression yields:

$$\hat{\mathbf{R}}_t^i(x_t^i, \tilde{x}_t^i) = \beta(t)\mathbf{R}_x(\tilde{x}_t^i, x_t^i) \sum_{x_0^i \in \Omega_x} p_{0|t}(x_0^i|G_t) \frac{\left[\exp(\int_0^t \beta(u)\mathbf{R}_x du)\right]_{x_0^i \tilde{x}_t^i}}{\left[\exp(\int_0^t \beta(u)\mathbf{R}_x du)\right]_{x_0^i x_t^i}}. \tag{53}$$

By replacing the dummy variable $\tilde{x}_t^i$ with the specific states x_s^i and x_t^i, the entries of the reverse rate matrix for nodes are obtained:

$$\hat{\mathbf{R}}_t^i(x_t^i, x_s^i) = \beta(t)\mathbf{R}_x(x_s^i, x_t^i) \sum_{x_0^i \in \Omega_x} p_{0|t}(x_0^i|G_t) \frac{\left[\exp(\int_0^t \beta(u)\mathbf{R}_x du)\right]_{x_0^i x_s^i}}{\left[\exp(\int_0^t \beta(u)\mathbf{R}_x du)\right]_{x_0^i x_t^i}}. \tag{54}$$

The same derivation applies to the edge transition rate matrix $\hat{\mathbf{R}}_t^{ij}$ for any edge (i, j), where $\Omega_e = \{1, 2, \cdots, b\}$:

$$\hat{\mathbf{R}}_t^{ij}(e_t^{ij}, e_s^{ij}) = \beta(t)\mathbf{R}_e(e_s^{ij}, e_t^{ij}) \sum_{e_0^{ij} \in \Omega_e} p_{0|t}(e_0^{ij}|G_t) \frac{\left[\exp(\int_0^t \beta(u)\mathbf{R}_e du)\right]_{e_0^{ij} e_s^{ij}}}{\left[\exp(\int_0^t \beta(u)\mathbf{R}_e du)\right]_{e_0^{ij} e_t^{ij}}}. \tag{55}$$

Following the same logic as the derivation of reverse transition probability in Appendix C.2, the factorized reverse probabilities are expressed in a unified form:

$$p_{s|t}(x_s^i|G_t) = \Big(\exp\Big(\Delta t \hat{\mathbf{R}}_t^i(x_t^i, x_t^i)\Big) - 1 + \delta(x_t^i, x_s^i)\Big) \frac{\hat{\mathbf{R}}_t^i(x_t^i, x_s^i)}{\hat{\mathbf{R}}_t^i(x_t^i, x_t^i)}, \tag{56}$$

$$p_{s|t}(e_s^{ij}|G_t) = \Big(\exp\Big(\Delta t \hat{\mathbf{R}}_t^i(e_t^{ij}, e_t^{ij})\Big) - 1 + \delta(e_t^{ij}, e_s^{ij})\Big) \frac{\hat{\mathbf{R}}_t^i(e_t^{ij}, e_s^{ij})}{\hat{\mathbf{R}}_t^i(e_t^{ij}, e_t^{ij})}. \tag{57}$$

C.5 Model Parameterization

Thus far, all derivations for the graph diffusion estimator are complete except for the posterior distributions $p_{0|t}(x_0^i|G_t)$ and $p_{0|t}(e_0^{ij}|G_t)$. A neural network ϕ_θ is employed to parameterize these distributions, denoted as $p_{0|t}^\theta(G_0|G_t)$. By taking the noisy graph G_t at time t as input, the model predicts the categorical probabilities for each node and edge attribute of the original clean graph:

$$\hat{x}_0^i = p_{0|t}^\theta(x_0^i|G_t), \quad \hat{e}_0^{ij} = p_{0|t}^\theta(e_0^{ij}|G_t). \tag{58}$$

The optimization of the network parameters θ is achieved by minimizing the following cross-entropy (CE) loss function:

$$\mathcal{L}_{DisCo} = \mathbb{E}_{\substack{t\sim\mathcal{U}(0,T) \\ \mathbf{G}_0\sim p_0 \\ \mathbf{G}_t\sim p_{t|0}}} \left[\sum_{i=1}^k \mathrm{CE}(x_0^i, \hat{x}_0^i) + \lambda \sum_{i,j=1}^k \mathrm{CE}(e_0^{ij}, \hat{e}_0^{ij}) \right]. \tag{59}$$

D DiGress Estimator (DiGress-E)

The estimator for $p_0(G_0)$ in discrete-time diffusion models can be derived as a specific case of Eq. (3) with a constant step size $\Delta t = 1$. This leads to the following formulation:

$$p_0(G_0) = \mathbb{E}_{\mathbf{G}_1, \mathbf{G}_2, \cdots, \mathbf{G}_T \sim p_{1,2,\cdots,T|0}} \left[p_T(\mathbf{G}_T) \prod_{t=1}^T \frac{p_{t-1|t}(\mathbf{G}_{t-1}|\mathbf{G}_t)}{p_{t|t-1}(\mathbf{G}_t|\mathbf{G}_{t-1})} \right]. \tag{60}$$

Similar to DisCo, DiGress [36] adopts a factorized transition kernel for the forward process. The transition probability from step $t-1$ to t is decomposed into independent components for nodes and edges:

$$p_{t|t-1}(G_t|G_{t-1}) = \prod_{i=1}^k p_{t|t-1}(x_t^i|x_{t-1}^i) \prod_{i,j=1}^k p_{t|t-1}(e_t^{ij}|e_{t-1}^{ij}). \tag{61}$$

In DiGress, the noise is injected via Markov transition matrices. The node-wise and edge-wise one-step transition probabilities are given by:

$$p_{t|t-1}(x_t^i|x_{t-1}^i) = [\mathbf{Q}_t^X]_{x_{t-1}^i x_t^i}, \tag{62}$$

$$p_{t|t-1}(e_t^{ij}|e_{t-1}^{ij}) = [\mathbf{Q}_t^E]_{e_{t-1}^{ij} e_t^{ij}}. \tag{63}$$

The transition matrices are structured to pull the distributions towards a prior distribution $\mathbf{m}$ (marginal distribution of the dataset):

$$\mathbf{Q}_t^X = \alpha_t \mathbf{I} + (1-\alpha_t)\mathbf{1}_a(\mathbf{m}^X)^\top \quad \text{and} \quad \mathbf{Q}_t^E = \alpha_t \mathbf{I} + (1-\alpha_t)\mathbf{1}_b(\mathbf{m}^E)^\top. \tag{64}$$

In these expressions, $\mathbf{1}_a \in \mathbb{R}^a$ and $\mathbf{1}_b \in \mathbb{R}^b$ are all-ones column vectors, while $\mathbf{m}^X \in \mathbb{R}^a$ and $\mathbf{m}^E \in \mathbb{R}^b$ are the marginal distributions (column vectors) of node types and edge types in the training set, respectively.

Due to the property of Markov Chain, the multi-step transition probability $p_{t|0}$ follows a similar form:

$$p_{t|0}(G_t|G_0) = \prod_{i=1}^k p_{t|0}(x_t^i|x_0^i) \prod_{i,j=1}^k p_{t|0}(e_t^{ij}|e_0^{ij}), \tag{65}$$

where the cumulative transition matrices $\bar{\mathbf{Q}}_t^X = \mathbf{Q}_1^X \mathbf{Q}_2^X \cdots \mathbf{Q}_t^X$ and $\bar{\mathbf{Q}}_t^E = \mathbf{Q}_1^E \mathbf{Q}_2^E \cdots \mathbf{Q}_t^E$ are as follows:

$$\bar{\mathbf{Q}}_t^X = \bar{\alpha}_t \mathbf{I} + (1 - \bar{\alpha}_t)\mathbf{1}_a(\mathbf{m}^X)^\top \quad \text{and} \quad \bar{\mathbf{Q}}_t^E = \bar{\alpha}_t \mathbf{I} + (1 - \bar{\alpha}_t)\mathbf{1}_b(\mathbf{m}^E)^\top. \tag{66}$$

The cumulative noise schedule follows a cosine schedule $\bar{\alpha}_t = \cos(0.5\pi(t/T + s)/(1 + s))^2$. As $t \to T$, the parameter $\bar{\alpha}_t$ approaches zero, which implies that the cumulative transition matrices $\bar{\mathbf{Q}}_t^X$ and $\bar{\mathbf{Q}}_t^E$ converge to $\mathbf{1}_a(\mathbf{m}^X)^\top$ and $\mathbf{1}_b(\mathbf{m}^E)^\top$, respectively. Consequently, the transition probability $p_{t|0}$ becomes independent of the initial state G_0, leading to the stationary distribution:

$$p_T(G_T) = \prod_{i=1}^{k} (\mathbf{m}^X)_{x_T^i} \prod_{i,j=1}^{k} (\mathbf{m}^E)_{e_T^{ij}}. \tag{67}$$

To obtain the reverse transition $p_{t-1|t}(G_{t-1}|G_t)$, the following Bayes' expansion is utilized:

$$\begin{aligned}
p_{t-1|t}(G_{t-1}|G_t) &= \sum_{G_0} p_{t-1,0|t}(G_{t-1}, G_0|G_t) && (68)\\
&= \sum_{G_0} p_{t-1|0,t}(G_{t-1}|G_0, G_t) \cdot p_{0|t}(G_0|G_t) && (69)\\
&= \sum_{G_0} \frac{p_{t-1,t|0}(G_{t-1}, G_t|G_0)}{p_{t|0}(G_t|G_0)} \cdot p_{0|t}(G_0|G_t) && (70)\\
&= \sum_{G_0} \frac{p_{t|t-1,0}(G_t|G_{t-1}, G_0) p_{t-1|0}(G_{t-1}|G_0)}{p_{t|0}(G_t|G_0)} \cdot p_{0|t}(G_0|G_t) && (71)\\
&= \sum_{G_0} \frac{p_{t|t-1}(G_t|G_{t-1}) p_{t-1|0}(G_{t-1}|G_0)}{p_{t|0}(G_t|G_0)} \cdot p_{0|t}(G_0|G_t). && (72)
\end{aligned}$$

The unknown posterior $p_{0|t}(G_0|G_t)$ is parameterized by a Graph Transformer $\phi_\theta(G_t)$, which predicts the clean graph $\hat{G}_0 = \{\hat{x}_0^i, \hat{e}_0^{ij}\}_{i,j=1}^k$, where the components are given by the corresponding outputs of the neural network: $\hat{x}_0^i = \phi_\theta(G_t)_i^X \in \mathbb{R}^a$ and $\hat{e}_0^{ij} = \phi_\theta(G_t)_{ij}^E \in \mathbb{R}^b$. Each component represents a categorical distribution (probability vector) over the possible node or edge types. The model is optimized using the cross-entropy loss between the true G_0 and the predicted $\hat{G}_0$:

$$\mathcal{L} = \mathbb{E}_{t\sim\mathcal{U}(1,T), \mathbf{G}_0\sim p_0, \mathbf{G}_t\sim p_{t|0}} \left[\sum_{i=1}^{k} \text{CE}(x_0^i, \hat{x}_0^i) + \lambda \sum_{i,j=1}^{k} \text{CE}(e_0^{ij}, \hat{e}_0^{ij}) \right]. \tag{73}$$

Once the model is trained, the estimated posterior probability for each node and each edge can be extracted from the corresponding entries of the predicted vectors:

$$p_{0|t}^\theta(x_0^i|G_t) = [\phi_\theta(G_t)_i^X]_{x_0^i} \quad \text{and} \quad p_{0|t}^\theta(e_0^{ij}|G_t) = [\phi_\theta(G_t)_i^E]_{e_0^{ij}}. \tag{74}$$

Consequently, the joint posterior $p_{0|t}^\theta(G_0|G_t)$ used in the reverse step is obtained through the product of individual probabilities:

$$p_{0|t}(G_0|G_t)^\theta = \prod_{i=1}^{k} p_{0|t}^\theta(x_0^i|G_t) \prod_{i,j=1}^{k} p_{0|t}^\theta(e_0^{ij}|G_t). \tag{75}$$

By substituting $p_{0|t}^\theta(G_0|G_t)$ back into the Bayes' expansion, the reverse transition $p_{t-1|t}(G_{t-1}|G_t)$ becomes computable, completing the DiGress estimator.

E DeFoG Estimator (DeFoG-E)

Unlike standard diffusion models, DeFoG is built upon the discrete flow matching framework [26, 4]. In this context, $p_0(G_0)$ represents the prior (noise) distribution, and $p_1(G_1)$ represents the data

distribution. Notably, the direction of time is defined such that $t = 0$ denotes pure noise and $t = 1$ denotes data. Consequently, the forward (nosing) process corresponds to decreasing time t ($1 \to 0$), while the reverse (denoising) process corresponds to increasing time t ($0 \to 1$). The objective in this section is to estimate the data density p_1 using a reverse-time trajectory:

$$p_1(G_1) = \mathbb{E}_{\mathbf{G}_{1-\Delta t}, \mathbf{G}_{1-2\Delta t}, \ldots, \mathbf{G}_0 \sim p_{1-\Delta t, 1-2\Delta t, \ldots, 0|1}} \left[p_0(\mathbf{G}_0) \prod_{t \in \mathcal{T}} \frac{p_{t+\Delta t|t}(\mathbf{G}_{t+\Delta t}|\mathbf{G}_t)}{p_{t|t+\Delta t}(\mathbf{G}_t|\mathbf{G}_{t+\Delta t})} \right], \tag{76}$$

where $\mathcal{T} = \{1 - \Delta t, 1 - 2\Delta t, \cdots, \Delta t, 0\}$. In this formulation, $p_{t|t+\Delta t}$ denotes the forward transition probability (noising) and $p_{t+\Delta t|t}$ denotes the reverse transition probability (denoising).

E.1 Forward Trajectory and Transition Probabilities

The noising trajectory in DeFoG is constructed by defining a probability path that linearly interpolates between the prior distribution p_0 and the data distribution p_1. For any $t \in [0, 1]$, the marginal distribution p_t is defined as:

$$p_t(z_t) = t p_1(z_t) + (1 - t) p_0(z_t). \tag{77}$$

To achieve this marginal distribution, the transition probability from the data z_1 to an intermediate state z_t is defined as:

$$p_{t|1}(z_t|z_1) = t\delta(z_t, z_1) + (1 - t) p_0(z_t). \tag{78}$$

To implement the estimator in Eq. (76), the forward transition probability $p_{t|s}$ for any $t < s$ must be determined. A general form for the transition probability $p_{t|s,1}$ is assumed as follows:

$$p_{t|s,1}(z_t|z_s, z_1) = \alpha(t, s)\delta(z_t, z_s) + \beta(t, s) p_0(z_t). \tag{79}$$

According to the law of total probability formula, the following identity must be satisfied:

$$\sum_{z_s} p_{t|s,1}(z_t|z_s, z_1) p_{s|1}(z_s|z_1) = p_{t|1}(z_t|z_1). \tag{80}$$

Substituting the assumed form of $p_{t|s,1}$ and the known expression of $p_{s|1}$ into the left-hand side (LHS) of the identity yields:

$$\begin{aligned}
\text{LHS} &= [\alpha(t, s)\delta(z_t, z_s) + \beta(t, s) p_0(z_t)] [s\delta(z_s, z_1) + (1 - s) p_0(z_s)] && (81)\\
&= \sum_{z_s} \alpha(t, s) s \delta(z_t, z_s)\delta(z_s, z_1) && (82)\\
&+ \sum_{z_s} \alpha(t, s)(1 - s)\delta(z_t, z_s) p_0(z_s) && (83)\\
&+ \sum_{z_s} \beta(t, s) s p_0(z_t)\delta(z_s, z_1) && (84)\\
&+ \sum_{z_s} \beta(t, s)(1 - s) p_0(z_t) p_0(z_s) && (85)\\
&= \alpha(t, s) s \delta(z_t, z_1) + [\alpha(t, s)(1 - s) + \beta(t, s)] p_0(z_t). && (86)
\end{aligned}$$

Comparing the coefficients of $\delta(z_t, z_1)$ and $p_0(z_t)$ with the right-hand side (RHS), $t\delta(z_t, z_1) + (1 - t) p_0(z_t)$, results in the following system of equations:

$$\begin{cases} \alpha(t, s) \cdot s = t \\ \alpha(t, s)(1 - s) + \beta(t, s) = 1 - t \end{cases}. \tag{87}$$

Solving this system gives $\alpha(t, s) = \dfrac{t}{s}$ and $\beta(t, s) = 1 - \dfrac{t}{s}$. The resulting expression for $p_{t|s,1}$ is thus:

$$p_{t|s,1}(z_t|z_s, z_1) = \frac{t}{s}\delta(z_t, z_s) + (1 - \frac{t}{s}) p_0(z_t). \tag{88}$$

Since this expression is independent of z_1, the transition probability $p_{t|s}$ is equivalent to $p_{t|s,1}$:

$$p_{t|s}(z_t|z_s) = \sum_{z_1} p_{t|s,1}(z_t|z_s, z_1)p_{1|s}(z_1|z_s) \tag{89}$$

$$= \sum_{z_1} [\frac{t}{s}\delta(z_t, z_s) + (1 - \frac{t}{s})p_0(z_t)]p_{1|s}(z_1|z_s) \tag{90}$$

$$= [\frac{t}{s}\delta(z_t, z_s) + (1 - \frac{t}{s})p_0(z_t)] \cdot \sum_{z_1} p_{1|s}(z_1|z_s) \tag{91}$$

$$= \frac{t}{s}\delta(z_t, z_s) + (1 - \frac{t}{s})p_0(z_t). \tag{92}$$

This confirms the Markovian property of the noising process. For nodes and edges, the transition probabilities within the noising trajectory are given by:

$$p_{t|t+\Delta t}(x_t^i|x_{t+\Delta t}^i) = \frac{t}{t+\Delta t}\delta(x_{t+\Delta t}^i, x_t^i) + (1 - \frac{t}{t+\Delta t})p_0(x_t^i), \tag{93}$$

$$p_{t|t+\Delta t}(e_t^{ij}|e_{t+\Delta t}^{ij}) = \frac{t}{t+\Delta t}\delta(e_{t+\Delta t}^{ij}, e_t^{ij}) + (1 - \frac{t}{t+\Delta t})p_0(e_t^{ij}). \tag{94}$$

The joint transition probability $p_{t|t+\Delta t}(G_t|G_{t+\Delta t})$ is then reconstructed as the product of these individual component-wise probabilities:

$$p_{t|t+\Delta t}(G_t|G_{t+\Delta t}) = \prod_i^k p_{t|t+\Delta t}(x_t^i|x_{t+\Delta t}^i) \prod_{i,j=1}^k p_{t|t+\Delta t}(e_t^{ij}|e_{t+\Delta t}^{ij}). \tag{95}$$

E.2 Reverse Process and Rate Matrix Construction

The reverse process $p_{t+\Delta t|t}(G_{t+\Delta t}|G_t)$ is assumed to be factorized across nodes and edges, implying that each component evolves according to its own transition rate:

$$p_{t+\Delta t|t}(G_{t+\Delta t}|G_t) = \prod_{i=1}^k p_{t+\Delta t}(x_{t+\Delta t}^i|G_t) \prod_{i,j=1}^k p_{t+\Delta t|t}(e_{t+\Delta t}^{ij}|G_t). \tag{96}$$

For a sufficiently small time step Δt, the transition probabilities can be approximated using the transition rates:

$$p_{t+\Delta t|t}(z_{t+\Delta t}|G_t) = \delta(z_t, z_{t+\Delta t}) + \mathbf{R}_t(z_t, z_{t+\Delta t})\Delta t, \tag{97}$$

where $\mathbf{R}_t$ denotes the time-dependent rate matrix. In the discrete flow matching framework, the rate matrix $\mathbf{R}_t$ is defined as the expectation of a conditional rate matrix $\mathbf{R}_{t|1}$ over the predicted data posterior $p_{1|t}(z_1|G_t)$:

$$\mathbf{R}_t(z_t, z_{t+\Delta t}) = \mathbb{E}_{p_{1|t}(z_1|G_t)}\left[\mathbf{R}_{t|1}(z_t, z_{t+\Delta t}|z_1)\right] \tag{98}$$

$$= \sum_{z_1} p_{1|t}(z_1|G_t)\mathbf{R}_{t|1}(z_t, z_{t+\Delta t}|z_1). \tag{99}$$

The conditional rate matrix $\mathbf{R}_{t|1}$ must satisfy the Kolmogorov forward equation:

$$\partial_t p_{t|1} = p_{t|1}\mathbf{R}_{t|1}. \tag{100}$$

To achieve this, DeFoG constructs $\mathbf{R}_{t|1}$ by combining a particular solution $\mathbf{R}_{t|1}^*$ with two additional terms, $\mathbf{R}_{t|1}^\eta$ and $\mathbf{R}_{t|1}^\omega$, both of which satisfy the detailed balance (DB) condition:

$$p_{t|1}(z_t|z_1)\mathbf{R}_{t|1}^{\text{DB}}(z_t, z_{t+\Delta t}|z_1) = p_{t|1}(z_{t+\Delta t}|z_1)\mathbf{R}_{t|1}^{\text{DB}}(z_{t+\Delta t}, z_t|z_1). \tag{101}$$

Specifically, $\mathbf{R}_{t|1}^*$ is defined to capture the primary probability flow:

$$\mathbf{R}_{t|1}^*(z_t, z_{t+\Delta t}|z_1) = \frac{\text{ReLU}[\partial_t p_{t|1}(z_{t+\Delta t}|z_1) - \partial_t p_{t|1}(z_t|z_1)]}{Z_t^{>0} p_{t|1}(z_t|z_1)}, \tag{102}$$

where $Z_t^{>0} = |\{z_t : p_{t|1}(z_t|z_1) > 0\}|$, $\delta(\cdot,\cdot)$ is the Kronecker delta function. Then, the terms $\mathbf{R}^{\eta}_{t|1}$ and $\mathbf{R}^{\omega}_{t|1}$ are formulated as specific instances of $\mathbf{R}^{\mathrm{DB}}_{t|1}$ to ensure stability and exploration:

$$\mathbf{R}^{\eta}_{t|1}(z_t, z_{t+\Delta t}|z_1) = \eta \cdot p_{t|1}(z_{t+\Delta t}|z_1), \tag{103}$$

$$\mathbf{R}^{\omega}_{t|1}(z_t, z_{t+\Delta t}|z_1) = \omega \cdot \frac{\delta(z_{t+\Delta t}, z_1)}{Z_t^{>0} p_{t|1}(z_t|z_1)}. \tag{104}$$

The total conditional rate matrix is thus the sum of these three components:

$$\mathbf{R}_{t|1}(z_t, z_{t+\Delta t}|z_1) = \mathbf{R}^{*}_{t|1}(z_t, z_{t+\Delta t}|z_1) + \mathbf{R}^{\eta}_{t|1}(z_t, z_{t+\Delta t}|z_1) + \mathbf{R}^{\omega}_{t|1}(z_t, z_{t+\Delta t}|z_1). \tag{105}$$

By substituting the probability path derivative $\partial_t p_{t|1}(z_t|z_1) = \delta(z_t, z_1) - p_0(z)$ and the conditional path $p_{t|1}(z|z_1) = t\delta(z_t, z_1) + (1-t)p_0(z_t)$ into the sum, the explicit form is obtained:

$$\mathbf{R}_{t|1}(z_t, z_{t+\Delta t}|z_1) = \frac{\mathrm{ReLU}[\partial_t p_{t|1}(z_{t+\Delta t}|z_1) - \partial_t p_{t|1}(z_t|z_1)]}{Z_t^{>0} p_{t|1}(z_t|z_1)} \tag{106}$$

$$+ \eta p_{t|1}(z_{t+\Delta t}|z_1) + \omega \frac{\delta(z_{t+\Delta t}, z_1)}{Z_t^{>0} p_{t|1}(z_t|z_1)} \tag{107}$$

$$= \frac{\mathrm{ReLU}[\delta(z_{t+\Delta t}, z_1) - \delta(z_t, z_1) - p_0(z_{t+\Delta t}) + p_0(z_t)] + \omega\delta(z_{t+\Delta t}, z_1)}{Z_t^{>0}[t\delta(z_t, z_1) + (1-t)p_0(z_t)]} \tag{108}$$

$$+ \eta \cdot [t\delta(z_{t+\Delta}, z_1) + (1-t)p_0(z_{t+\Delta t})], \tag{109}$$

where p_0 represents the prior distribution (taken as the marginal distribution of the dataset in DeFoG). The terms η and ω are two adjustable hyperparameters.

By applying the definition of the expected rate matrix $\mathbf{R}_t$ (Eq. (98)) and the specific form of $\mathbf{R}_{t|1}$ (Eq. (106)) to each component of the graph, the final transition rates for nodes x^i and edges e^{ij} are expressed as:

$$\mathbf{R}^i_t(x^i_t, x^i_{t+\Delta t}) = \sum_{x^i_1} p_{1|t}(x^i_1|G_t) \left[\mathbf{R}^i_{t|1}(x^i_t, x^i_{t+\Delta t}|x^i_1)\right], \tag{110}$$

$$\mathbf{R}^{ij}_t(e^{ij}_t, e^{ij}_{t+\Delta t}) = \sum_{e^{ij}_1} p_{1|t}(e^i_1|G_t) \left[\mathbf{R}^{ij}_{t|1}(e^{ij}_t, e^{ij}_{t+\Delta t}|e^{ij}_1)\right], \tag{111}$$

$$\mathbf{R}^i_{t|1}(x^i_t, x^i_{t+\Delta t}|x^i_1) = \frac{\mathrm{ReLU}[\partial_t p_{t|1}(x^i_{t+\Delta t}|x^i_1) - \partial_t p_{t|1}(x^i_t|x^i_1)]}{Z_t^{>0} p_{t|1}(x^i_t|x^i_1)} \tag{112}$$

$$+ \eta p_{t|1}(x^i_{t+\Delta t}|x^i_1) + \omega \frac{\delta(x^i_{t+\Delta t}, x^i_1)}{Z_t^{>0} p_{t|1}(x^i_t|x^i_1)} \tag{113}$$

$$= \frac{\mathrm{ReLU}[\delta(x^i_{t+\Delta t}, x^i_1) - \delta(x^i_t, x^i_1) - p_0(x^i_{t+\Delta t}) + p_0(x^i_t)] + \omega\delta(x^i_{t+\Delta t}, x^i_1)}{Z_t^{>0}[t\delta(x^i_t, x^i_1) + (1-t)p_0(x^i_t)]} \tag{114}$$

$$+ \eta \cdot [t\delta(x^i_{t+\Delta}, x^i_1) + (1-t)p_0(x^i_{t+\Delta t})], \tag{115}$$

$$\mathbf{R}^{ij}_{t|1}(e^{ij}_t, e^{ij}_{t+\Delta t}|e^{ij}_1) = \frac{\mathrm{ReLU}[\partial_t p_{t|1}(e^{ij}_{t+\Delta t}|e^{ij}_1) - \partial_t p_{t|1}(e^{ij}_t|e^{ij}_1)]}{Z_t^{>0} p_{t|1}(e^{ij}_t|e^{ij}_1)} \tag{116}$$

$$+ \eta p_{t|1}(e^{ij}_{t+\Delta t}|e^{ij}_1) + \omega \frac{\delta(e^{ij}_{t+\Delta t}, e^{ij}_1)}{Z_t^{>0} p_{t|1}(e^{ij}_t|e^{ij}_1)} \tag{117}$$

$$= \frac{\mathrm{ReLU}[\delta(e^{ij}_{t+\Delta t}, e^{ij}_1) - \delta(e^{ij}_t, e^{ij}_1) - p_0(e^{ij}_{t+\Delta t}) + p_0(e^{ij}_t)] + \omega\delta(e^{ij}_{t+\Delta t}, e^{ij}_1)}{Z_t^{>0}[t\delta(e^{ij}_t, e^{ij}_1) + (1-t)p_0(e^{ij}_t)]} \tag{118}$$

$$+ \eta \cdot [t\delta(e^{ij}_{t+\Delta}, e^{ij}_1) + (1-t)p_0(e^{ij}_{t+\Delta t})]. \tag{119}$$

While the above expressions provide the exact form of the transition rates, they remain dependent on the unknown data posterior $p_{1|t}(G_1|G_t)$. In practice, this posterior cannot be computed analytically for complex graph distributions. To address this, DeFoG employs a Graph Transformer $\phi_\theta(G_t)$ to approximate the categorical distributions of the clean graph. The network is trained to predict the probabilities for each node and edge, denoted as $\hat{x}_1^i = \phi_\theta(G_t)_i^X$ and $\hat{e}_1^{ij} = \phi_\theta(G_t)_{ij}^E$, respectively. The optimization is guided by a cross-entropy loss:

$$\mathcal{L}_{\text{DeFoG}} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\mathbf{G}_1\sim p_1,\mathbf{G}_t\sim p_{t|1}}\left[\sum_{i=1}^{k}\text{CE}(x_1^i,\hat{x}_1^i) + \lambda\sum_{i,j=1}^{k}\text{CE}(e_1^{ij},\hat{e}_1^{ij})\right]. \tag{120}$$

The sampling process $\mathbf{G}_t \sim p_{t|1}$ in the loss function is governed by the factorized probability path. Specifically, $p_{t|1}(G_t|G_1)$ defines how a clean graph is progressively diffused into the prior distribution, and is expressed as the product of independent component-wise transitions:

$$p_{t|1}(G_t|G_1) = \prod_i^k p_{t|1}(x_t^i|x_1^i)\prod_{i,j=1}^{k} p_{t|1}(e_t^{ij}|e_1^{ij}). \tag{121}$$

The component-wise probability paths are defined as:

$$p_{t|1}(x_t^i|x_1^i) = t\delta(x_t^i,x_1^i) + (1-t)p_0(x_t^i), \tag{122}$$

$$p_{t|1}(e_t^{ij}|e_1^{ij}) = t\delta(e_t^{ij},e_1^{ij}) + (1-t)p_0(e_t^{ij}). \tag{123}$$

Once the model ϕ_θ is trained, the posterior distributions $p_{1|t}(x_1^i|G_t)$ and $p_{1|t}(e_1^{ij}|G_t)$ are replaced by the network's predicted probabilities $p_{1|t}^\theta(x_1^i|G_t)$ and $p_{1|t}^\theta(e_1^{ij}|G_t)$, respectively. Consequently, the expected rate matrices $\mathbf{R}_t^i$ in Eq. (110) and $\mathbf{R}_t^{ij}$ in Eq. (111) can be computed using the explicit forms derived above. These rates in turn determine the transition probabilities $p_{t+\Delta t|t}(x_{t+\Delta t}^i|G_t)$ and $p_{t+\Delta t|t}(e_{t+\Delta t}^{ij}|G_t)$ in Eq. (96), thereby completing the estimation of $p_G(G_1)$ in Eq. (76).

F Subgraph Sampling Methods

F.1 Acceptance-Rejection Sampling (ARS)

The Acceptance-Rejection Sampling (ARS) is the most straightforward method for subgraph sampling from a graph set $\mathcal{G}$. As shown in Alg. 4, this algorithm first samples a graph $G^{(i)}$ from $\mathcal{G}$ with probability proportional to its edge count. Then, k nodes are selected uniformly at random from $G^{(i)}$ to form an induced subgraph. If the induced subgraph is connected, it is accepted; otherwise, it is rejected.

While theoretically unbiased, ARS suffers from extreme inefficiency due to its high rejection rate. As k increases, the probability of forming a connected subgraph decreases rapidly, making the algorithm computationally impractical for large subgraph sizes.

F.2 Neighbour Reservoir Sampling (NRS)

The Neighbor Reservoir Sampling (NRS) algorithm [19] is designed to enhance sampling efficiency by directly exploring the neighborhood of the current node set. As shown in Alg. 5, the algorithm proceeds in two main stages. It first expands from a randomly sampled edge until the node set reaches size k, and then adopts a Reservoir Sampling mechanism to iteratively update the candidate nodes.

Although NRS is efficient, it should be noted that the sampling process is theoretically biased. Despite this bias, its high efficiency and ability to maintain connectivity make it a practical and robust choice for large-scale subgraph sampling tasks.

F.3 Rand-ESU

The Rand-ESU algorithm [39] is a randomized version of the Exact Subgraph Enumeration (ESU) method. As shown in Alg. 6, it systematically explores a graph by constructing an enumeration tree

Algorithm 4 Acceptance-Rejection Sampling (ARS)

Input: Raw graph set $\mathcal{G} = \{G^{(1)}, G^{(2)}, \cdots, G^{(|\mathcal{G}|)}\}$, subgraph size k, target count tc.
Output: Subgraph dataset $\mathcal{D}_k$.
function ARS($\mathcal{G}$, k, tc)
    Initialize empty set $\mathcal{D}_k = \emptyset$
    Calculate weights $w_i \leftarrow \dfrac{|E(G^{(i)})|}{\sum_{j=1}^{|\mathcal{G}|} |E(G^{(j)})|}$ // $|E(G^{(i)})|$ is the number of edges for $G^{(i)}$
    $\mathbf{w} \leftarrow (w_1, w_2, ..., w_{|\mathcal{G}|})$
    while $|\mathcal{D}_k| < tc$ **do**
        Sample $i \sim$ Categorical($\mathbf{w}$)
        $G \leftarrow G^{(i)}$
        Select k nodes from G at random
        Generate the induced subgraph g
        if g is connected **then**
            $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{g\}$
        end if
    end while
    return $\mathcal{D}_k$
end function

of depth k. At each level d of the tree, the algorithm decides whether to continue the traversal with probability $p_d \in (0, 1]$. The mechanism allows for controlling the number of samples through the probability list $p_{1:k}$, where a lower p_d significantly reduces the computational overhead in exchange for a smaller sample set. The algorithm is theoretically unbiased because each subgraph of size k is associated with a unique path in the enumeration tree, ensuring an equal sampled probability for all subgraphs at the same depth.

F.4 Rand-FaSE

The Rand-FaSE algorithm [25] is a randomized subgraph sampling method based on an ESU-style enumeration framework.

As shown in Alg. 7, it explores each input graph by constructing an enumeration tree of depth k, where nodes correspond to partial subgraphs and edges represent incremental extensions. At each depth d of the tree, the traversal is continued with probability $p_d \in (0, 1]$, which enables explicit control over the sampling rate through the probability list $p_{1:(k-1)}$. Smaller values of p_d reduce the computational cost by probabilistically pruning the enumeration tree, at the expense of a smaller set of sampled subgraphs.

During traversal, Rand-FaSE organizes expansion paths using a trie structure indexed by LS-labels, allowing subgraphs with identical structural extensions to be aggregated online. For each sampled subgraph of size k, the algorithm records the cumulative sampling probability along its enumeration path and applies inverse-probability weighting at the leaf level. As a result, Rand-FaSE produces unbiased frequency estimates for subgraphs while maintaining efficient exploration of the enumeration tree.

G Further Implementation Details

This section presents more details about the datasets, evaluation protocol, and hyperparameters.

G.1 Datasets Description

NAS-Bench-101. The NAS-Bench-101 dataset [43] is a classic open-source benchmark built by the Google Brain team to provide a reproducible and comparable standard for Neural Architecture Search (NAS). The dataset contains a total of 423,624 unique cell-based architectures, each with a maximum of 7 nodes. These architectures are composed of 5 different operator types.

Algorithm 5 Neighbour Reservoir Sampling (NRS)

Input: Raw graph set $\mathcal{G} = \{G^{(1)}, G^{(2)}, \cdots, G^{(|\mathcal{G}|)}\}$, subgraph size k, target count tc.
Output: Subgraph dataset $\mathcal{D}_k$.
function NRS($\mathcal{G}$, k, tc)
    Initialize empty set $\mathcal{D}_k = \emptyset$
    Calculate weights $w_i \leftarrow \dfrac{|E(G^{(i)})|}{\sum_{j=1}^{|\mathcal{G}|} |E(G^{(j)})|}$ // $|E(G^{(i)})|$ is the number of edges for $G^{(i)}$
    $\mathbf{w} \leftarrow (w_1, w_2, ..., w_{|\mathcal{G}|})$
    while $|\mathcal{D}_k| < tc$ **do**
        Sample $i \sim$ Categorical($\mathbf{w}$)
        $G \leftarrow G^{(i)}$
        $g \leftarrow$ NRSonSingleGraph(G, k)
        $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{g\}$
    end while
    return $\mathcal{D}_k$
end function

function NRSonSingleGraph(G, k)
    Sample an edge $e \in E(G)$ at random
    $V \leftarrow \{x \in e\}$
    while $|V| < k$ **do**
        $EL \leftarrow \{(x, y) \in E(G) : (x \in V \wedge y \notin V) \vee (x \notin V \wedge y \in V)\}$
        Sample $e \in EL$ at random
        $V \leftarrow V \cup \{x \in e \mid x \notin V\}$
    end while
    $V_u \leftarrow V(G) - V$ // the unprocessed nodes of G
    $EL \leftarrow \{(x, y) : (x \in V \wedge y \in V_u) \vee (x \in V_u \wedge y \in V)\}$
    $i \leftarrow k$
    while $|EL| > 0$ **do**
        $i \leftarrow i + 1$
        Sample $e \in EL$ at random
        $v \leftarrow \{x \in e \mid x \in V_u\}$
        $V_u \leftarrow V_u - \{v\}$
        $\alpha \sim$ Uniform$(0, 1)$
        if $\alpha < \dfrac{k}{i}$ **then**
            Sample $u \in V$ at random
            $V' \leftarrow (V - \{u\}) \cup \{v\}$
            $g' \leftarrow$ the induced subgraph of V'
            if g' is connected **then**
                $V \leftarrow V'$
            end if
        end if
        $EL \leftarrow \{(x, y) : (x \in V \wedge y \in V_u) \vee (x \in V_u \wedge y \in V)\}$
    end while
    $g \leftarrow$ the induced subgraph of V
    return g
end function

Algorithm 6 Rand-ESU

Input: Raw graph set $\mathcal{G} = \{G^{(1)}, G^{(2)}, \cdots, G^{(|\mathcal{G}|)}\}$, subgraph size k, target count tc, probability list $p_{1:k}$.
Output: Subgraph dataset $\mathcal{D}_k$.

Notation:
$N(V', G) := \{u \in V(G) - V' \mid \exists v \in V', (u, v) \in E(G) \vee (v, u) \in E(G)\}$
$N_{excl}(w, V_{Sub}, G) := \{u \in N(\{w\}, G) \mid u \notin V_{Sub} \cup N(V_{Sub}, G)\}$

function Rand-ESU($\mathcal{G}$, k, tc)
    Initialize empty set $\mathcal{D}_k = \emptyset$
    while $|\mathcal{D}_k| < tc$ **do**
        for $i = 1 \to |\mathcal{G}|$ **do**
            $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup$ EnumerateSubgraphs(G, k)
        end for
    end while
    $\mathcal{D}_k \leftarrow$ Select tc elements from $\mathcal{D}_k$ at random
    return $\mathcal{D}_k$
end function

function EnumerateSubgraphs(G, k)
    Initialize $\mathcal{D}_{single} = \emptyset$
    for each $v \in V(G)$ **do**
        $V_{Ext} \leftarrow \{u \in N(\{v\}, G) : u > v\}$
        With probability p_1, **call** ExtendSubgraph($\{v\}, V_{Ext}, v, G, k, \mathcal{D}_{single}$)
    end for
    return $\mathcal{D}_{single}$
end function

function ExtendSubgraph($V_{Sub}, V_{Ext}, v, G, k, \mathcal{D}_{single}$)
    if $|V_{Sub}| = k$ **then**
        $\mathcal{D}_{single} \leftarrow \mathcal{D}_{single} \cup \{$the induced subgraph of $V_{Sub}\}$
        return
    end if
    while $V_{Ext} \neq \emptyset$ **do**
        Remove an arbitrarily chosen node w from V_{Ext}
        $V'_{Ext} \leftarrow V_{Ext} \cup \{u \in N_{excl}(w, V_{Sub}, G) : u > v\}$
        $d \leftarrow |V_{Sub}| + 1$ // The depth of the next potential call
        With probability p_d, **call** ExtendSubgraph($V_{Sub} \cup \{w\}, V'_{Ext}, v, G, k, \mathcal{D}_{single}$)
    end while
end function

Algorithm 7 Rand-FaSE

Input: Graph set $\mathcal{G} = \{G^{(1)}, \ldots, G^{(|\mathcal{G}|)}\}$, subgraph size k, target count tc, probability list $p_{1:k}$.
Output: Subgraph dataset $\mathcal{D}_k$.

Notation:
$N(V', G) := \{u \in V(G) - V' \mid \exists v \in V', (u, v) \in E(G) \vee (v, u) \in E(G)\}$
$N_{excl}(w, V_{Sub}, G) := \{u \in N(\{w\}, G) \mid u \notin V_{Sub} \cup N(V_{Sub}, G)\}$
$\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}, \ell)$ (FaSE trie, which is labeld by LS-label)

function Rand-FaSE($\mathcal{G}$, k, tc)
    Initialize empty set $\mathcal{D}_k = \emptyset$, trie $\mathcal{T}$ with root, probability list $p_{1:(k-1)}$
    while $|\mathcal{D}_k| < tc$ **do**
        for $i = 1 \rightarrow |\mathcal{G}|$ **do**
            EnumerateSubgraphs($G^{(i)}$, k, $p_{1:(k-1)}$, $\mathcal{T}$)
        end for
        $\mathcal{D}_k \leftarrow$ Aggregate from leaves of $\mathcal{T}$
    end while
    $\mathcal{D}_k \leftarrow$ Select tc elements from $\mathcal{D}_k$ at random
    return $\mathcal{D}_k$
end function

function EnumerateSubgraphs(G, k, $p_{1:(k-1)}$, $\mathcal{T}$)
    for each $v \in V(G)$ **do**
        $V_{Sub} \leftarrow \{v\}$; $V_{Ext} \leftarrow \{u \in N(\{v\}, G) : u > v\}$
        $d \leftarrow 1$; $q \leftarrow 1$
        Insert LS-label of v into trie $\mathcal{T}$
        call ExtendSubgraph(G, k, V_{Sub}, V_{Ext}, v, $\mathcal{T}$, d, q)
    end for
end function

function ExtendSubgraph(G, k, V_{Sub}, V_{Ext}, v, $\mathcal{T}$, d, q)
    $q \leftarrow q \cdot p_d$
    if $|V_{Sub}| = k$ **then**
        $weight \leftarrow 1/q$ // Cumulative sampling probability
        $\sigma \leftarrow \mathrm{Canon}(G[V_{Sub}])$ //σ: Canonical signature
        $\mathcal{T}[\sigma]$.count $+= weight$
        return
    end if
    while $V_{Ext} \neq \emptyset$ **do**
        Remove a node w from V_{Ext}
        With probability p_d:
            $\ell \leftarrow \mathrm{LS}(w, V_{Sub})$ // LS-label
            $u \leftarrow \mathcal{T}_{\mathrm{child}}(u, \ell)$
            $V'_{Ext} \leftarrow V_{Ext} \cup \{u \in N_{excl}(w, V_{Sub}, G) : u > v\}$
            call ExtendSubgraph(G, k, $V_{Sub} \cup \{w\}$, V'_{Ext}, v, $\mathcal{T}$, $d + 1$, q)
    end while
end function

NAS-Bench-201. The NAS-Bench-201 dataset [8] is another cell-based benchmark with 15,625 unique architectures. In this paper, the original edge-based operator representation is converted to a node-based representation, following the method in DiNAS [1]. After this conversion, each cell contains 8 nodes, and the architectures are composed of 7 distinct operator types.

NAS-Bench-301. NAS-Bench-301 [47] is a surrogate-based benchmark for NAS, designed to address the high computational cost of evaluating architectures in a huge search space. Following the setting in DiNAS [1], this paper collected 10,000 architectures from the DARTS search space. In this benchmark, each architecture contains 11 nodes. The architectures are composed of 11 different operator types.

NAS-Bench-NLP. The NAS-Bench-NLP dataset [15] is a benchmark focused on Recurrent Neural Network (RNN) cells for natural language processing tasks. Following the setting in DiNAS [1], this paper filters this dataset to contain a total of 7258 architectures. In this benchmark, each architecture contains at most 13 nodes. The architectures are composed of 9 different operator types.

Younger. Unlike the previously discussed synthetic benchmarks, the Younger [42] dataset is a large-scale, diverse collection of real-world neural architectures. The maximum number of nodes for one architecture in the dataset is about 50000, with a total of 14,230 unique architectures, composed of 314 operator types.

HighSchool. The HighSchool dataset [20] is a social network benchmark that captures interactions between students in a high school setting. It consists of a single graph containing 329 nodes, representing the contact network of the school. In terms of node attributes, the original dataset includes multiple features such as student class and gender. To align with the tasks in this paper and ensure a sufficient diversity of node types, class and gender are combined into a unified representation, resulting in a total of 21 distinct node types.

PROTEINS. The PROTEINS dataset [3] is a widely used bioinformatics benchmark representing protein structures as graphs. The dataset comprises a total of 1113 unique graphs. The largest graph in the dataset contains 620 nodes. The nodes are categorized into 3 distinct node types, which represent the different types of secondary structures.

Statistical information about these datasets are shown in Table 3.

Table 3: Statistical information of different datasets. Statistics include the maximum number of nodes in a single graph, the total number of nodes across all graphs, and the number of node types.

Dataset	**Max#Nodes**	**Sum#Nodes**	**#Node Types**
NAS-Bench-101	7	2898k	5
NAS-Bench-201	8	125k	7
NAS-Bench-301	11	220k	11
NAS-Bench-NLP	13	83k	9
Younger	45670	28780k	314
HighSchool	329	329	21
PROTEINS	620	43k	3

G.2 Evaluation Protocol

The evaluation protocol consists of two primary components: the assessment of the Graph Diffusion Estimator's predictive performance and the evaluation of the search framework's overall effectiveness in identifying frequent subgraphs.

Evaluation of the Graph Diffusion Estimator. The performance of the estimator is quantified using Spearman's ρ and Kendall's τ. These metrics evaluate the monotonic relationship between the predicted probabilities and the ground-truth frequencies.

For a specific subgraph size k on a given dataset, the ground-truth frequency F_i, for each non-isomorphic subgraph class i, is obtained via exhaustive enumeration. The graph diffusion estimator provides a predicted probability P_i for each class.

- For the graph diffusion estimator, P_i represents the direct output of the model.

- For sampling-based baselines, P_i is defined as the frequency of class i in the sampled set. If a class is not sampled, P_i is set to 0.

Given two sequences of predicted $\mathbf{P}$ and ground-truth frequencies $\mathbf{F}$, both of length n, the coefficients are defined as follows:

- Spearman's ρ: Let $R(P_i)$ and $R(F_i)$ denote the ranks of the predicted probability and the ground-truth frequency within their respective sequences. The coefficient is defined as:

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} (R(P_i) - R(F_i))^2}{n(n^2 - 1)} \tag{124}$$

- Kendall's τ: Let n_c and n_d denote the number of concordant and discordant pairs, respectively. For any pair of observations (P_i, F_i) and (P_j, F_j), the pair is concordant if their ranks follow the same order (i.e., $(R(P_i) - R(P_j))(R(F_i) - R(F_j)) > 0$), and discordant otherwise. The coefficient is defined as:

$$\tau = \frac{2(n_c - n_d)}{n(n-1)} \tag{125}$$

Evaluation of the Overall Framework. The effectiveness of the overall framework is assessed by its capacity to discover frequent subgraphs. The VF2 algorithm, as implemented in the *igraph* library, is employed to perform exact subgraph isomorphism counting to provide the ground-truth counts. The execution time of VF2 increases exponentially as the subgraph size grows, becoming computationally expensive for larger patterns. To ensure computational feasibility within a reasonable timeframe, a cutoff time is established for the VF2 algorithm. Subgraph counts are only recorded if their occurrences are found before the cutoff time. The performance of the framework is evaluated using the mean and median of the actual counts of the top-N subgraphs identified by the framework.

- For the search framework, the top-N subgraphs correspond to the K subgraphs with the highest predicted probabilities from the graph diffusion estimator.
- For the sampling-based methods, the top-N are the N most frequent subgraphs observed in the sampled dataset.

G.3 Baselines

The experiments incorporate six baselines: four sampling-based methods (ARS, NRS, Rand-ESU, and Rand-FaSE), one VAE-based model (GraphVAE), and one supervised GNN-based estimator (GNNRanker).

For sampling-based methods, performance is measured according to the evaluation protocol in Section G.2. For GraphVAE, the encoder produces the mean and variance of the latent distribution for each subgraph G_0. By sampling from the latent space to reconstruct the graph, the Evidence Lower Bound (ELBO) is computed as a surrogate for the log-generative probability. By averaging ELBOs over multiple latent space samplings, a generative probability is obtained from the perspective of GraphVAE. This value corresponds to $p_0(G_0)$ in graph diffusion estimators, allowing GraphVAE to be assessed using the same evaluation protocol.

GNNRanker employs a GIN encoder to predict subgraph frequency scores. Its supervision is derived from empirical subgraph counts in the Rand-ESU samples, transformed by $\log(1 + count)$ and standardized using training-set statistics. The model is trained with a combination of Smooth L1 regression loss and pairwise ranking loss.

G.4 Model Architecture

The graph diffusion estimator employs a graph-to-graph neural network ϕ_θ as its backbone to predict the distribution of nodes and edges. Given an input graph characterized by node attributes $X \in \mathbb{R}^{n \times a}$ and edge attributes $E \in \mathbb{R}^{n \times n \times b}$, the model outputs transformed features in the same dimensions.

The architecture is based on the GraphTransformer introduced in DiGress [36]. The input attributes first pass through an MLP to map them into a latent space. Then, several XEyTransformerLayers iteratively update node and edge attributes. Finally, the output MLP projects these latent representations to the required output dimensions.

Table 4: The number of sampled subgraphs required to reach a sampling density of 0.1 across different datasets and sampling methods.

Dataset	Subgraph Size $k = 4$				Subgraph Size $k = 5$			
	ARS	NRS	Rand-ESU	Rand-FaSE	ARS	NRS	Rand-ESU	Rand-FaSE
NAS-Bench-101	399	404	568	422	14642	14293	16165	14807
NAS-Bench-201	286	279	273	286	1316	1320	1323	1335
NAS-Bench-301	3126	2892	3013	2797	27588	25032	28335	26912
NAS-Bench-NLP	1697	2076	1768	1778	8511	9218	8212	7475
Younger	X	140316	171903	96692	X	108010	152780	87000

To adapt to Directed Acyclic Graphs (DAGs), this study introduces a positional encoding mechanism that is not present in the original DiGress backbone. Specifically, node levels are calculated based on the topological structure as:

$$\text{level}(v) = \max_{u \in \mathcal{N}_{in}(v)} (\text{level}(u) + 1) \tag{126}$$

Nodes are then sorted according to these levels. After the initial node attributes X are projected into the latent space via the first MLP layer, absolute positional encodings (PE) derived from this topological ordering are added to the latent representations. This modification incorporates the hierarchical information of DAGs into the model.

G.5 Hardware Setting

Experiments were run with Intel Xeon E5-2620 v3 CPU, 63GB RAM, and NVIDIA RTX-2080Ti GPUs.

G.6 Hyperparameters

For the generative models DisCo [41], DeFoG [26], and DiGress [36], the hyperparameters are primarily kept consistent with the original configurations reported in their respective papers. However, specific adjustments are made to meet the requirements of this study. For DisCo, the number of sampling steps is set to 100; α and γ are set to 0.8 and 2, respectively. For DeFoG, the time distortion for the reverse process is configured as *polydec*. The number of estimation rounds is set to 20 for DisCo and DiGress, and 60 for DeFoG.

For the sampling methods, ARS and NRS are parameter-free. For Rand-ESU and Rand-FaSE, the sampling probability at depth d for a target subgraph size k is determined by $p_d = (1 - d/(k+1))^r$. The parameter r controls the sampling density and is assigned based on the dataset and subgraph size. For NAS-Bench-101, r is set to 1. For NAS-Bench-201, NAS-Bench-301, and NAS-Bench-NLP, r is set to 0. For the Younger dataset, if $k \leq 12$, r is set to 3, if $13 \leq k \leq 15$, r is set to 2.5. The cutoff time for VF2 algorithm is set to 20 minutes, with the counting process parallelized across 10 CPU cores.

H Additional Experimental Results

H.1 The Number of Sampled Subgraphs

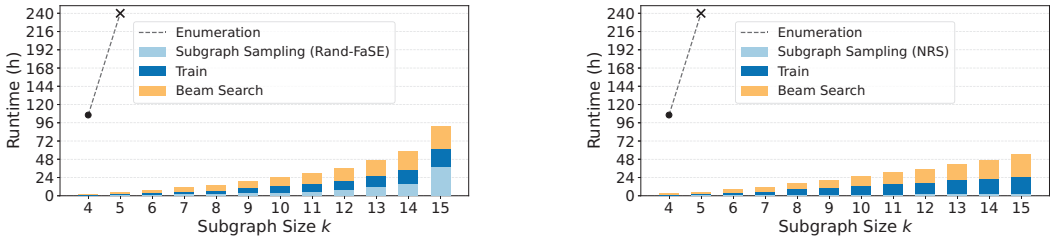
Table 4 reports the number of sampled subgraphs required to reach a sampling density of 0.1 across different datasets and sampling methods.

H.2 Runtime Analysis

This section evaluates the computational efficiency of the GraDE framework, specifically focusing on the time required for model training and the frequent subgraph discovery process. Table 5 reports the training time for DisCo-E, DeFoG-E, and DiGress-E across various datasets, with the number of training epochs fixed at 200, the sampling method set to Rand-ESU, and the sampling density set to 0.1.

Table 5: Training time (hours) of different estimators. The sampling method is set to Rand-ESU, and the sampling density is set to 0.1.

Dataset	Subgraph size $k = 4$			Subgraph size $k = 5$		
	DisCo-E	DeFoG-E	DiGress-E	DisCo-E	DeFoG-E	DiGress-E
NAS-Bench-101	0.10	0.09	0.08	0.35	0.20	0.18
NAS-Bench-201	0.10	0.06	0.07	0.12	0.07	0.07
NAS-Bench-301	0.19	0.08	0.13	0.55	0.58	0.42
NAS-Bench-NLP	0.14	0.11	0.07	0.27	0.22	0.24
Younger	5.97	2.03	1.81	5.26	2.16	2.11



(a) Runtime combination of GraDE (Rand-FaSE).

(b) Runtime combination of GraDE (NRS).

Figure 8: Runtime combination of GraDE.

The results indicate that training on NAS-Bench-101, NAS-Bench-201, NAS-Bench-301, and NAS-Bench-NLP is highly efficient, with total training times consistently under one hour. On the Younger dataset, training requires more time. This is primarily due to the larger variety of operator types (314) in the Younger dataset architecture space and the increased number of samples needed to maintain a 0.1 sampling density.

The runtime characteristics of GraDE using Rand-FaSE and NRS are illustrated in Figure 8a and Figure 8b, respectively. For GraDE(Rand-FaSE), the runtime trend is similar to Rand-ESU: the sampling part takes a larger share as k increases, nearly equaling the training and search time at $k = 15$. In terms of absolute runtime, Rand-FaSE is generally faster than Rand-ESU. In contrast, the sampling time for GraDE(NRS) is so short that it is almost negligible compared to the training and search phases. Since the time for training and searching in GraDE is relatively stable, GraDE(NRS) provides a faster trade-off for users who prioritize speed over discovery capability. These results further demonstrate the computational feasibility of the GraDE framework.

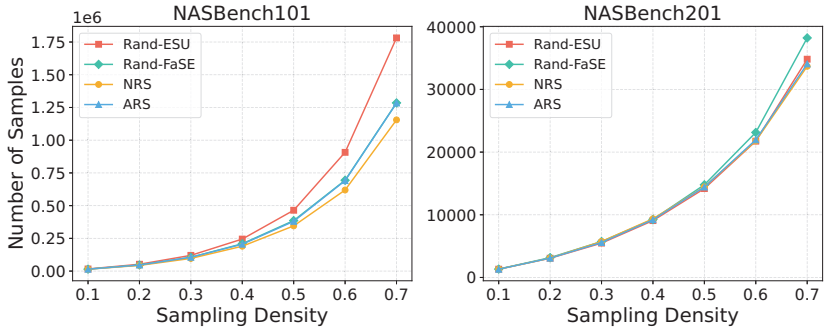


Figure 9: The number of sampled subgraphs vs. sampling density. These results are obtained using Rand-ESU, Rand-FaSE, NRS, and ARS on the NAS-Bench-101 and NAS-Bench-201 datasets.

H.3 Impact of Sampling Density on Sampling Cost

To provide empirical evidence for the relationship between sampling density and sampling cost, an analysis of the required sample sizes was conducted. This evaluation employs Rand-ESU, Rand-FaSE, NRS, and ARS across the NAS-Bench-101 and NAS-Bench-201 datasets. The sampling density in these experiments ranges from 0.1 to 0.7.

Figure 9 illustrates the number of sampled subgraphs as a function of sampling density across both datasets. The results consistently demonstrate that, regardless of the sampling method employed, the total number of required samples grows at an exponential rate as the density increases. This observation confirms that maintaining a high sampling density leads to prohibitive computational costs.

H.4 Comparison of Different Methods on Various Datasets

This section provides a comprehensive evaluation of various frequency estimators across a diverse range of datasets and sampling methods. In addition to the NAS-Bench-101, NAS-Bench-201, NAS-Bench-301, and Younger datasets discussed in the main text, the experiments here further include results on NAS-Bench-NLP.

Moreover, while Table 1 in the main text only presents the performance of different generative models using a single sampling strategy (Rand-ESU), the tables in this section provide a more detailed analysis by showcasing the effectiveness of these methods across all four sampling strategies. For all evaluations, Spearman's ρ and Kendall's τ are reported for subgraph sizes $k \in \{4, 5\}$ with a consistent sampling density of 0.1.

Note that due to the large scale of the Younger dataset, the ARS method is computationally infeasible. Therefore, all corresponding entries for ARS in Table 10 are marked with an "X" to indicate the absence of results.

The full results are presented in Tables 6, 7, 8, 9, and 10. Regarding general performance, DisCo-E achieves the best results under most conditions across the NAS-Bench-101, NAS-Bench-201, NAS-Bench-NLP, and Younger datasets, significantly outperforming sampling-based methods. On NAS-Bench-301, the performance rankings change depending on the subgraph size k. For $k = 4$, DeFoG-E gets the best results in most cases, while DisCo-E and DiGress-E often take the second-best spots. When the subgraph size increases to $k = 5$, DiGress-E becomes the top performer, and DisCo-E usually keeps the second-best position.

The experimental results show that DisCo-E has good generality, as it works well with different types of datasets and different sampling methods. Specifically, for the DisCo-E estimator, the Rand-ESU sampling strategy consistently gives the best or near-best performance across all NAS datasets compared to other sampling strategies. This is also true for the Younger dataset, where the combination of DisCo-E and Rand-ESU achieves the best results. Consequently, taking all conditions into account, DisCo-E paired with Rand-ESU provides the most stable and reliable performance overall.

Table 6: Comparison of different methods with various sampling strategies on NAS-Bench-101.

Size	Estimator	ARS		NRS		Rand-ESU		Rand-FaSE	
		$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$
4	Sampling-only	0.332	0.273	0.349	0.287	0.388	0.319	0.331	0.272
	GraphVAE	0.379	0.255	0.412	0.281	0.378	0.256	0.383	0.259
	DiGress-E	0.545	0.376	0.346	0.232	0.692	0.501	0.621	0.436
	DeFoG-E	0.694	0.499	0.783	0.575	0.827	0.653	0.836	0.654
	DisCo-E	**0.869**	**0.686**	**0.872**	**0.688**	**0.860**	**0.681**	**0.877**	**0.699**
5	Sampling-only	0.426	0.366	0.422	0.362	0.442	0.379	0.418	0.358
	GraphVAE	0.391	0.276	0.399	0.282	0.424	0.303	0.426	0.305
	DiGress-E	0.831	0.671	0.842	0.682	0.877	0.725	0.833	0.666
	DeFoG-E	0.864	0.698	0.865	0.699	0.899	0.741	0.862	0.694
	DisCo-E	**0.896**	**0.749**	**0.898**	**0.750**	**0.938**	**0.809**	**0.885**	**0.726**

Table 7: Comparison of different methods with various sampling strategies on NAS-Bench-201.

Size	Estimator	ARS		NRS		Rand-ESU		Rand-FaSE	
		$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$
4	Sampling-only	0.400	0.373	0.386	0.360	0.402	0.375	0.417	0.389
	GraphVAE	0.581	0.460	0.449	0.351	0.255	0.195	0.402	0.314
	DiGress-E	0.676	0.540	0.742	0.561	0.791	0.592	0.812	0.654
	DeFoG-E	0.795	0.641	**0.820**	0.662	**0.815**	0.620	0.748	0.604
	DisCo-E	**0.859**	**0.717**	0.819	**0.677**	0.778	**0.641**	**0.851**	**0.709**
5	Sampling-only	0.306	0.279	0.308	0.279	0.332	0.302	0.296	0.269
	GraphVAE	0.683	0.544	0.667	0.529	0.701	0.562	0.714	0.573
	DiGress-E	**0.868**	**0.723**	0.868	0.725	0.845	0.698	0.766	0.611
	DeFoG-E	0.552	0.423	0.554	0.427	0.607	0.471	0.490	0.372
	DisCo-E	0.845	0.700	**0.873**	**0.731**	**0.884**	**0.746**	**0.772**	**0.622**

Table 8: Comparison of different methods with various sampling strategies on NAS-Bench-301.

Size	Estimator	ARS		NRS		Rand-ESU		Rand-FaSE	
		$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$
4	Sampling-only	0.418	0.347	0.417	0.346	0.428	0.356	0.431	0.357
	GraphVAE	0.179	0.117	0.201	0.134	0.220	0.146	0.277	0.179
	DiGress-E	0.789	**0.622**	0.119	0.073	0.673	0.493	0.802	0.632
	DeFoG-E	**0.797**	0.598	**0.797**	**0.602**	0.545	0.395	**0.852**	**0.664**
	DisCo-E	0.755	0.553	0.777	0.578	**0.740**	**0.544**	0.774	0.578
5	Sampling-only	0.421	0.362	0.414	0.356	0.423	0.364	0.428	0.368
	GraphVAE	0.460	0.349	0.470	0.356	0.462	0.347	0.486	0.367
	DiGress-E	**0.805**	**0.632**	**0.803**	**0.630**	**0.816**	**0.648**	0.795	**0.626**
	DeFoG-E	0.730	0.551	0.732	0.551	0.714	0.537	0.736	0.560
	DisCo-E	0.793	0.606	0.783	0.596	0.789	0.600	**0.798**	0.616

Table 9: Comparison of different methods with various sampling strategies on NAS-Bench-NLP.

Size	Estimator	ARS		NRS		Rand-ESU		Rand-FaSE	
		$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$	$\rho\uparrow$	$\tau\uparrow$
4	Sampling-only	0.429	0.368	0.434	0.373	0.433	0.371	0.425	0.364
	GraphVAE	0.493	0.360	0.474	0.346	0.466	0.338	0.459	0.332
	DiGress-E	0.543	0.403	0.283	0.204	0.398	0.290	0.469	0.342
	DeFoG-E	0.570	0.421	0.610	0.456	0.569	0.424	0.590	0.440
	DisCo-E	**0.677**	**0.515**	**0.678**	**0.517**	**0.674**	**0.513**	**0.679**	**0.518**
5	Sampling-only	0.406	0.365	0.411	0.369	0.400	0.360	0.383	0.343
	GraphVAE	0.433	0.326	0.438	0.330	0.433	0.326	0.433	0.326
	DiGress-E	0.536	0.410	0.572	0.442	0.584	0.452	0.519	0.397
	DeFoG-E	0.564	0.432	0.577	0.443	0.568	0.437	0.557	0.427
	DisCo-E	**0.637**	**0.496**	**0.624**	**0.485**	**0.660**	**0.517**	**0.621**	**0.482**

Table 10: Comparison of different methods with various sampling strategies on Younger.

Size	Estimator	ARS		NRS		Rand-ESU		Rand-FaSE	
		$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$	$\rho \uparrow$	$\tau \uparrow$
4	Sampling-only	X	X	<u>0.454</u>	<u>0.380</u>	<u>0.458</u>	<u>0.384</u>	<u>0.350</u>	<u>0.290</u>
	GraphVAE	X	X	0.375	0.262	0.378	0.264	0.301	0.209
	DiGress-E	X	X	0.358	0.251	0.374	0.262	0.113	0.077
	DeFoG-E	X	X	0.367	0.259	0.415	0.293	0.285	0.198
	DisCo-E	X	X	**0.556**	**0.402**	**0.558**	**0.404**	**0.423**	**0.299**
5	Sampling-only	X	X	0.414	<u>0.347</u>	0.416	<u>0.349</u>	0.320	<u>0.267</u>
	GraphVAE	X	X	0.331	0.231	0.320	0.223	0.264	0.182
	DiGress-E	X	X	0.314	0.218	0.392	0.275	0.051	0.035
	DeFoG-E	X	X	<u>0.453</u>	0.320	<u>0.470</u>	0.333	<u>0.343</u>	0.239
	DisCo-E	X	X	**0.547**	**0.394**	**0.559**	**0.404**	**0.440**	**0.311**

H.5 Estimator Performance for Larger Subgraphs

To evaluate estimator performance beyond $k \in \{4, 5\}$, a controlled subset of Younger is constructed by randomly selecting 250 graphs with at most 50 nodes. Their connected $k = 8$ subgraphs are exhaustively enumerated to obtain ground-truth frequencies. This setting enables direct ranking evaluation at a larger subgraph size while keeping enumeration computationally feasible. All sampling methods use a sampling density of 0.1, and the GraDE estimators are trained on Rand-ESU samples at the same density.

Table 11: Estimator performance on a controlled Younger subset at $k = 8$ and sampling density 0.1. "X" indicates that ARS exceeds the evaluation time limit.

Model	Class	$\rho \uparrow$	$\tau \uparrow$
ARS	Sampling	X	X
NRS	Sampling	0.098	0.089
Rand-ESU	Sampling	0.259	0.239
Rand-FaSE	Sampling	0.185	0.170
DiGress-E	Diffusion	0.347	0.261
DeFoG-E	Flow	<u>0.354</u>	<u>0.266</u>
DisCo-E	Diffusion	**0.378**	**0.288**

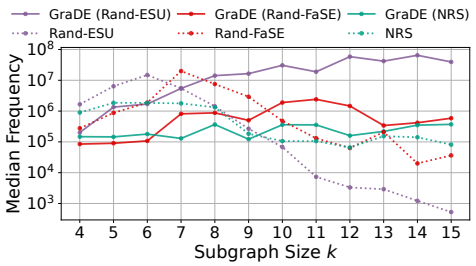
As shown in Table 11, all three GraDE estimators outperform the completed sampling baselines in both ranking metrics. DisCo-E achieves the best performance, supporting the effectiveness of the graph diffusion estimator for larger subgraphs.

H.6 Discovery Capability of the GraDE Framework

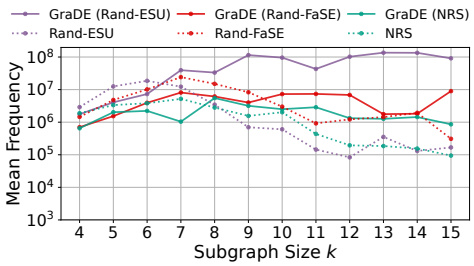
This section provides an extended analysis of the discovery performance of GraDE, supplementing the results discussed in Section 5.4. Due to space constraints in the main text, only the median frequencies were highlighted. Therefore, a more comprehensive statistical evaluation is presented here. Figure 10a includes the previously reported median frequencies for completeness, while Figure 10b illustrates the mean frequencies of the top-50 discovered subgraphs.

As shown in Figure 10a and Figure 10b, a performance crossover is observed as k increases. While pure sampling-based methods exhibit better performance for smaller k, GraDE begins to gain a substantial advantage as k grows, especially for $k \geq 10$. In this range ($k \geq 10$), GraDE (Rand-ESU) outperforms Rand-ESU by an average of over $100\times$ across both mean and median frequencies. Similarly, GraDE (Rand-FaSE) and GraDE (NRS) achieve over $5\times$ and $2.5\times$ average improvements over their respective sampling-based methods.

Notably, GraDE(Rand-ESU) emerges as the best configuration starting from $k = 8$. As k further increases, the gap between GraDE (Rand-ESU) and the best sampling-based methods widens signifi-



(a) Median frequencies of top-50 discovered subgraphs.



(b) Mean frequencies of top-50 discovered subgraphs.

Figure 10: Median and mean frequencies of top-50 discovered subgraphs. GraDE(Rand-ESU), GraDE(Rand-FaSE), and GraDE(NRS) denote the GraDE framework with corresponding sampling methods.

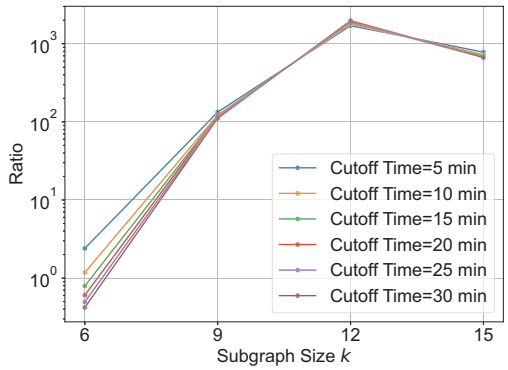
cantly. Specifically, in the large-scale scenarios ($k \geq 10$), this advantage exceeds $30\times$ across all cases. These results demonstrate that the GraDE framework possesses a substantial advantage in discovering large-scale frequent subgraph patterns, effectively mitigating the limitations of sampling-based methods in massive search spaces.

H.7 Impact of Cutoff Time

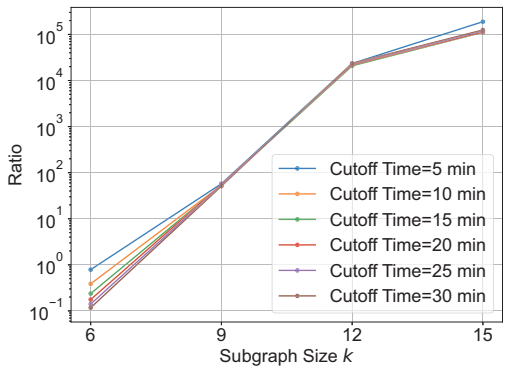
This section investigates the sensitivity of the evaluation results to the cutoff time employed in the VF2 algorithm. Since exact frequency counting for large subgraphs is often computationally prohibitive, this analysis determines whether the relative performance between methods remains consistent under different cutoff times.

The experiment evaluates the top 50 subgraphs discovered by GraDE (with Rand-ESU) and those found by the sampling-based method (Rand-ESU). The mean and median frequencies of these subgraphs are calculated using the VF2 algorithm across six different cutoff times: 5, 10, 15, 20, 25, and 30 minutes. Figure 11 illustrates the ratio of the frequencies (GraDE vs. sampling-based method) across different subgraph sizes $k \in \{6, 9, 12, 15\}$.

The results indicate that the frequency ratios across all six cutoff time settings are remarkably consistent, particularly as the subgraph size increases. This stability suggests that while absolute frequency counts may be affected by the cutoff time, the relative frequency advantage of the subgraphs discovered by GraDE remains unchanged. Therefore, a 20-minute cutoff time is considered a reasonable and reliable threshold for comparing the performance of different methods.



(a) Mean frequency ratio.



(b) Median frequency ratio.

Figure 11: Sensitivity analysis of frequency ratios across different VF2 cutoff times (5–30 min) on the Younger dataset. Each line represents a specific cutoff time across various subgraph sizes $k \in \{6, 9, 12, 15\}$.

H.8 Further Analysis on Beam Search

To further evaluate the robustness of the discovery framework, this section investigates the impact of beam width B (representing the top-N candidates retained at each step) and the effectiveness of the guided pruning strategy. Performance is measured by the top-50 median frequency of discovered subgraph patterns.

Sensitivity to Beam Width. As shown in Table 12, increasing the beam width B consistently enhances the quality of discovered subgraphs. This performance gain becomes more pronounced as the subgraph size k increases (e.g., achieving a 2.4× improvement at $k = 8$ when B increases from 250 to 1000), suggesting that a larger beam width is beneficial for the effectiveness of GraDE framework.

Table 12: Top-50 median frequency across different beam widths B.

$B \backslash k$	**4**	**5**	**6**	**7**	**8**
250	1.9e5	2.4e5	1.0e6	4.2e6	5.9e6
500	2.2e5	1.2e6	1.4e6	5.2e6	6.0e6
1000	2.0e5	1.3e6	1.7e6	5.4e6	1.4e7

Pruning Risk and Initialization. Standard GraDE is compared against "Fork" paths, where the top-B beam seeds are replaced by B random samples from the non-top-B set at $k = 3$ or $k = 6$. Table 13 reveals that standard GraDE significantly outperforms both forked paths. Notably, the sharp decline in frequency for Fork@k=3 (representing random initialization) highlights the necessity of starting from high-frequency patterns. Furthermore, the superiority of standard GraDE over Fork@k=6 justifies the pruning strategy: while pruning carries inherent risks of discarding potential candidates, the retained set consistently provides a higher-quality foundation for subsequent expansion.

Table 13: Comparison between standard GraDE and "Fork" paths.

Path $\backslash k$	**4**	**5**	**6**	**7**	**8**
Fork@$k = 3$	1.1e3	3.4e3	1.5e4	1.5e4	8.8e5
Fork@$k = 6$	-	-	-	3.4e6	9.1e6
Standard GraDE	2.0e5	1.3e6	1.7e6	5.4e6	1.4e7

H.9 Case Study: From Controllable Discovery to Practical Applications

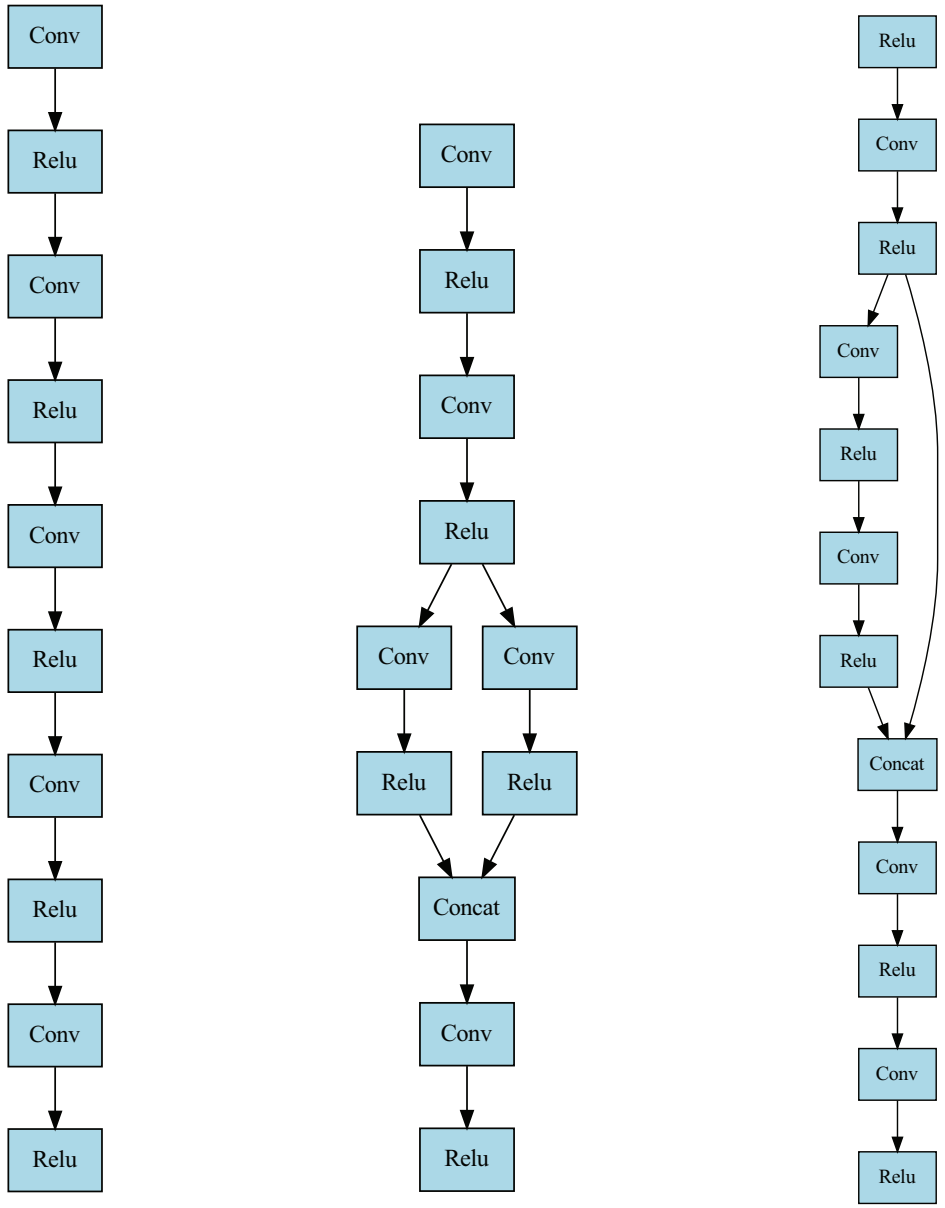
The incremental nature of the GraDE framework allows for the constrained search, offering significant efficiency and controllability over sampling-based methods. In GraDE, domain-specific constraints can be integrated as pruning rules during the expansion process. Specifically, any candidate subgraph that violates the predefined constraints is discarded immediately, preventing the search from exploring uninterested branches. In contrast, sampling-based methods can only perform post-hoc filtering, where uninterested structures are removed only after the entire sampling process is finished.

This section considers two common structural constraints: (1) a Node-type Constraint, requiring the subgraph to contain at least one convolutional operation; and (2) an Edge Connectivity Constraint, requiring the in-degree and out-degree of every node in the subgraph to be at most 2. Table 14 reports the proportion of subgraphs that simultaneously satisfy both constraints among the raw samples collected on the Younger dataset by various sampling-based methods across different subgraph sizes k.

Table 14: Proportion of interested samples satisfying both node-type and connectivity constraints.

Size k	4	5	6	7	8	9	10
NRS	3.4150%	3.8375%	4.1500%	3.9800%	3.9950%	3.8500%	4.0075%
Rand-ESU	2.5800%	1.1950%	0.4175%	0.1275%	0.0350%	0.0200%	0.0025%
Rand-FaSE	15.2550%	13.4950%	8.4250%	3.7575%	1.7525%	0.6900%	0.2900%

As shown in Table 14, the proportion of interested samples obtained by sampling-based methods is remarkably low, especially as the subgraph size increases. This highlights the extreme sparsity of constrained structures in the overall architecture space, making post-hoc filtering highly inefficient. In contrast, GraDE guarantees that 100% of the discovered subgraphs satisfy the predefined constraints, as uninterested candidates are pruned at the earliest possible stage of the search process. As shown in Figure 12, three typical cases are discovered by the aforementioned constrained GraDE framework. These frequent structures can serve as building blocks for neural network construction, which helps to accelerate the overall design process.



(a) Case A (b) Case B (c) Case C

Figure 12: Discovered subgraph patterns by constrained GraDE framework on Younger.

To further evaluate the practical utility of the discovered patterns, the typical cases in Figure 12 are implemented as PyTorch modules. These implementations strictly map the structural topologies identified by GraDE to concrete functional blocks, as shown in the following code:

```
class CaseAModule(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv4 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv5 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        return x

class CaseBModule(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv4 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv5 = nn.Conv2d(128, 64, kernel_size=3, padding=1, bias=False)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        split_input = F.relu(self.conv2(x))
        branch_l = F.relu(self.conv3(split_input))
        branch_r = F.relu(self.conv4(split_input))
        concat = torch.cat([branch_l, branch_r], dim=1)
        out = F.relu(self.conv5(concat))
        return out

class CaseCModule(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)
        self.conv4 = nn.Conv2d(128, 64, kernel_size=3, padding=1, bias=False)
        self.conv5 = nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False)

    def forward(self, x):
        top_relu = F.relu(x)
        re_before_split = F.relu(self.conv1(top_relu))
        bl1_r = F.relu(self.conv2(re_before_split))
        bl2_r = F.relu(self.conv3(bl1_r))
        cat = torch.cat([bl2_r, re_before_split], dim=1)
        re4 = F.relu(self.conv4(cat))
        res = F.relu(self.conv5(re4))
        return res
```

The computational efficiency of these modules is evaluated using TVM v0.18.0 [6] on an NVIDIA RTX 2080Ti GPU. For each case, the input resolution is 256, the number of input channels is 64, the kernel size is 3, and the batch size is 32. Three optimization strategies are compared: (1) Unoptimized, representing the raw graph execution without any TVM optimization; (2) Unfused, where operators are optimized independently; (3) Manual Fusion, where specific structures like Conv and ReLU are manually combined before undergoing TVM's operator-level optimization; and (4) Automated Fusion, where the entire subgraph is treated as a single block for TVM's global optimization.

The experimental results, summarized in Table 15, reveal several key insights. First, optimization is essential; compared to the unoptimized baseline, all TVM-optimized strategies achieve significant speedups across all cases. Second, subgraph-level optimization is essential, as both fusion-based strategies consistently outperform the unfused baseline. For instance, in Case B, manual fusion achieves a $1.27\times$ speedup over the unfused version, reducing execution latency from 151.56 ms to 119.54 ms. Third, manual fusion often yields superior results compared to automated fusion. In Case B, manual fusion results in 119.54 ms latency, whereas automated fusion only reaches 139.77 ms, with similar trends observed in other cases.

These findings demonstrate that while automated compilers are powerful, significant performance gains can be achieved through manual intervention in specific structural patterns. Since manual optimization involves substantial costs, it is most practical to apply it to the most critical and frequent structures. Consequently, the ability of GraDE to identify these frequent subgraphs provides a vital foundation for system optimization, ensuring manual efforts are focused where they yield the highest impact.

Table 15: Inference latency (ms) under different TVM optimization strategies.

Subgraph	Unoptimized	Unfused	Manual Fusion	Automated Fusion	Max Speedup
Case A (Fig. 12a)	135.06	106.16	96.18	98.72	1.40×
Case B (Fig. 12b)	162.21	151.56	119.54	139.77	1.36×
Case C (Fig. 12c)	164.60	129.49	112.42	124.83	1.46×

I Similar but Distinct Work

To further clarify the positioning of GraDE, this section distinguishes the framework from recent learning-based approaches that exhibit superficial similarities but address different research goals. SPMiner [44] employs a node-anchored frequency definition and is restricted to unattributed graphs, whereas GraDE targets graph-level frequency and supports attributed graphs. MotiFiesta [23] considers subgraphs with minor structural variations as the same motif, which deviates from the rigorous exact isomorphism required in this work. Micro-Graph [48] similarly relies on approximate motifs rather than exact structural isomorphism. Another work [35] focuses on small subgraphs ($k \in \{3, 4\}$), lacking the scalability to handle larger subgraphs.

J Impact Statement

This work focuses on the efficient discovery of frequent subgraphs within neural network architectures. By utilizing graph diffusion models to score subgraph frequencies instead of relying on exponential subgraph counting, the proposed method transforms an NP-hard task into a computationally feasible one. These discovered neural network motifs facilitate optimizing execution efficiency, accelerating architecture design, and uncovering novel structural insights. Given its scientific focus, this work presents no foreseeable negative social impacts or ethical risks.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope. The main claims can be found in Abstract and Introduction sections.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of the work are discussed in the Conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: For all theoretical results, this paper provides corresponding proofs, with the most essential proofs presented in the main text and the others in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The core experimental setups and hyperparameters are provided in the Experiments section. Further details are provided in Further Implementation Details and Additional Experimental Results sections in appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code and a representative dataset are provided in the supplemental material. Other experimental datasets are all publicly available and were omitted from the supplemental material solely due to their volume.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: These details are specified in the Experiments section of the main text, as well as in the Further Implementation Details and Additional Experimental Results sections in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main experiments were conducted over multiple runs, with both the mean and standard deviation reported to demonstrate statistical significance. By adhering to evaluation protocols and reporting results across various settings in both the Experiments and Appendix, the paper ensures that the findings faithfully and accurately reflect the method's performance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper analyzes execution time in the Experiments section and provides comprehensive details regarding computing resources and runtime in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: See Impact Statement section in appendix.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper ensures that all utilized code, models, and datasets are correctly attributed to their respective original papers through proper citations.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: New assets introduced in the paper, such as code, are well documented. The documentation is provided alongside the assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: Core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.