PARD: Permutation-invariant Autoregressive Diffusion for Graph Generation

Lingxiao Zhao
Carnegie Mellon University
lingxiaozlx@gmail.com

Xueying Ding
Carnegie Mellon University
xding2@andrew.cmu.edu

Leman Akoglu
Carnegie Mellon University
lakoglu@andrew.cmu.edu

Abstract

Graph generation has been dominated by autoregressive models due to their simplicity and effectiveness, despite their sensitivity to node ordering. Diffusion models, on the other hand, have garnered increasing attention as they offer comparable performance while being permutation-invariant. Current graph diffusion models generate graphs in a one-shot fashion, however they require extra features and thousands of denoising steps to achieve optimal performance. We introduce PARD, a Permutation-invariant AutoRegressive Diffusion model that integrates diffusion models with autoregressive methods. PARD harnesses the effectiveness and efficiency of the autoregressive model while maintaining permutation invariance without order sensitivity. Specifically, we show that contrary to sets, elements in a graph are not entirely unordered and there is a unique partial order for nodes and edges. With this partial order, PARD generates a graph in a block-by-block, autoregressive fashion, where each block's probability is conditionally modeled by a shared diffusion model with an equivariant network. To ensure efficiency while being expressive, we further propose a higher-order graph transformer, which integrates transformer with PPGN. Like GPT, we extend the higher-order graph transformer to support parallel training of all blocks. Without any extra features, PARD achieves state-of-the-art performance on molecular and non-molecular datasets, and scales to large datasets like MOSES containing 1.9M molecules. PARD is open-sourced at https://github.com/LingxiaoShawn/Pard

1 Introduction

Graphs provide a powerful abstraction for representing relational information in many domains, including social networks, biological and molecular structures, recommender systems, and networks of various infrastructures such as computers, roads, etc. Accordingly, generative models of graphs that learn the underlying graph distribution from data find applications in network science [7], drug discovery [26, 42], protein design [1, 43], and various use-cases for Internet of Things [10]. Importantly, they serve as a prerequisite for building a generative foundation model [6] for graphs.

Despite significant progress in generative models for images and language, graph generation is uniquely challenged by its inherent combinatorial nature. Specifically: 1) Graphs are naturally high-dimensional and *discrete* with *varying sizes*, contrasting with the continuous space and fixed-size advancements that cannot be directly applied here; 2) Being permutation-invariant objects, graphs require modeling an *exchangeable probability* distribution, where permutations of nodes and edges do not alter a graph's probability; and 3) The rich substructures in graphs necessitate an expressive model capable of capturing *higher-order* motifs and interactions. Several graph generative models have been proposed to address (part of) these challenges, based on various techniques like autoregression [48, 28], VAEs [37], GANs [11], flow-based methods [36], and denoising diffusion [32, 44]. Among these, autoregressive models and diffusion models stand out with superior performance, thus significant popularity. However, current autoregressive models, while

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

https://doi.org/10.52202/079017-0229

efficient, are sensitive to node/edge order with non-exchangeable probabilities; whereas diffusion models, though promising, are less efficient, requiring thousands of denoising steps and extra node/edge/graph-level features (structural and/or domain-specific) to achieve high generation quality.

In this paper, we introduce PARD (leopard in Ancient Greek), the *first* Permutation-invariant AutoRegressive Diffusion model that combines the efficiency of autoregressive methods and the quality of diffusion models together, while retaining the property of exchangeable probability. Instead of generating an entire graph directly, we explore the direction of generating through *block-wise* graph enlargement. Graph enlargement offers a fine-grained control over graph generation, which can be particularly advantageous for real-world applications that require local revisions to generate graphs. Moreover, it essentially decomposes the joint distribution of the graph into a series of simpler conditional distributions, thereby leveraging the data efficiency characteristic of autoregressive modeling. We also argue that graphs, unlike sets, inherently exhibit a *unique partial order* among nodes, naturally facilitating the decomposition of the joint distribution. Thanks to this unique partial order, PARD's block-wise autoregressive sequence is permutation-invariant, unlike any prior graph autoregressive methods in the literature.

To model the conditional distribution of nodes and edges within a block, we have, for the first time, identified a fundamental challenge in equivariant models for generation: it is impossible for *any* equivariant model, no matter how powerful, to perform general graph transformations without symmetry breaking. However, through a diffusion process that injects noise, a permutation equivariant network can progressively denoise to realize targeted graph transformations. This approach is inspired by the annealing process where energy is initially heightened before achieving a stable state, akin to the process of tempering iron. Our analytical findings naturally lead to the design of our proposed PARD that combines autoregressive approach with local block-wise discrete denoising diffusion. Using a diffusion model with equivariant networks ensures that each block's conditional distribution is exchangeable. Coupled with the permutation-invariant block sequence, this renders the entire process permutation-invariant and the joint distribution exchangeable. What is more, this inevitable combination of autoregression and diffusion successfully combines the strength of both approaches while getting rid of their shortcomings: By being permutation-invariant, a key advantage of diffusion, it generalizes better than autoregressive models while also being much more data-efficient. By decomposing the challenging joint probability into simpler conditional distributions, an advantage of autoregression, it requires significantly fewer diffusion steps, outperforming pure diffusion methods by a large margin. Additionally, each inference step in the diffusion process incurs lower computational cost by processing only the generated part of the graph, rather than the entire graph. And it can further leverage caching mechanisms (to be explored in future) to avoid redundant computations.

Within PARD, we further propose several architectural improvements. First, to achieve 2-FWL expressivity with improved memory efficiency, we propose a higher-order graph transformer that integrates transformer with PPGN [30], while utilizing a significantly reduced representation size for edges. Second, to ensure training efficiency without substantial overhead compared to the original diffusion model, we design a GPT-like causal mechanism to support parallel training of all blocks. These extensions are generalizable and can lay the groundwork for a higher-order GPT.

PARD achieves new SOTA performance on many molecular and non-molecular datasets *without any extra features*, significantly outperforming DiGress [44]. Thanks to efficient architecture and parallel training, PARD scales to large datasets like MOSES [33] with 1.9M graphs. Finally, not only PARD can serve as a generative foundation model for graphs in the future, its autoregressive parallel mechanism can further be combined with language models for language-graph generative pretraining.

2 Related Work

Autoregressive (AR) Models for Graph Generation. AR models create graphs step-by-step, adding nodes and edges sequentially. This method acknowledges graphs' discrete nature but faces a key challenge as there is no inherent order in graph generation. To address this, various strategies have been proposed to simplify orderings and approximate the marginalization over permutations; i.e. $p(G) = \sum_{\pi \in \mathcal{P}(G)} p(G, \pi)$. Li et al. [27] propose using random or deterministic empirical orderings. GraphRNN [48] aligns permutations with breadth-first-search (BFS) ordering, with a many-to-one mapping. GRAN [28] offers marginalization over a family of canonical node orderings, including node degree descending, DFS/BFS tree rooted at the largest degree node, and k-core ordering. GraphGEN [16] uses a single canonical node ordering, but does not guarantee the same

https://doi.org/10.52202/079017-0229

canonical ordering during generation. Chen et al. [9] avoid defining ad-hoc orderings by modeling the conditional probability of orderings, $p(\pi|G)$, with a trainable AR model, estimating marginalized probabilities during training to enhance both the generative model and the ordering probability model.

Diffusion Models for Graph Generation. EDP-GNN [32] is the first work that adapts score matching [39] to graph generation, by viewing graphs as matrices with continuous values. GDSS [24] generalizes EDP-GNN by adapting SDE-based diffusion [40] and considers node and edge features. Yan et al. [46] argues that learning exchangeable probability with equivariant networks is hard, hence proposes permutation-sensitive SwinGNN with continuous-state score matching. Previous works apply continuous-state diffusion to graph generation, ignoring the natural discreteness of graphs. DiGress [44] is the first to apply discrete-state diffusion [3, 20] to graph generation and achieves significant improvement. However, DiGress relies on many additional structural and domain-specific features. GraphArm [25] applies Autoregressive Diffusion Model (ADM) [21] to graph generation, where exactly one node and its adjacent edges decay to the absorbing states at each forward step based on a *random* node order. Similar to AR models, GraphArm is permutation sensitive.

We remark that although both are termed "autoregressive diffusion", it is important to distinguish that PARD is *not* ADM. The term "autoregressive diffusion" in our context refers to the integration of autoregressive methods with diffusion models. In contrast, ADM represents a specific type of discrete denoising diffusion where exactly one dimension decays to an absorbing state at a time in the forward diffusion process. See Fan et al. [13] for a survey of recent diffusion models on graphs.

3 Permutation-invariant Autoregressive Denoising Diffusion

We first introduce setting and notations. We focus on graphs with categorical features. Let $G = (\mathcal{V}, \mathcal{E})$ be a *labeled* graph with the number of distinct node and edge labels denoted K_v and K_e, respectively. Let $\boldsymbol{v}^i \in \{0,1\}^{K_v}, \forall i \in \mathcal{V}$ be the one-hot encoding of node i's label. Let $\boldsymbol{e}^{i,j} \in \{0,1\}^{K_e}, \forall i,j \in \mathcal{V}$ be the one-hot encoding of the label for the edge between node i and j. We also represent "absence of edge" as a type of edge label, hence $|\mathcal{E}| = |\mathcal{V}| \times |\mathcal{V}|$. Let $\mathrm{V} \in \{0,1\}^{|\mathcal{V}| \times K_v}$ and $\mathrm{E} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}| \times K_e}$ be the collection of one-hot encodings of all nodes and edges using the *default node order*, and let $\mathrm{G} := (\mathrm{V}, \mathrm{E})$. To describe probability, let $\mathbf{x}$ be a random variable with its sampled value $\boldsymbol{x}$. Similarly, $\mathbf{G}$ is a random graph with its sampled graph G. In diffusion process, noises are injected from t=0 to t=T with T being the maximum time step. Let $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$ be the random variable of observed data with underlying distribution $p_{\text{data}}(\mathbf{x}_0)$, $\mathbf{x}_t \sim q(\mathbf{x}_t)$ be the random variable at time t, and let $\mathbf{x}_{t|s} \sim q(\mathbf{x}_t|\mathbf{x}_s)$ denote the conditional random variable. Also, we interchangeably use $q(\mathbf{x}_t|\mathbf{x}_s)$, $q(\mathbf{x}_t{=}\boldsymbol{x}_t|\mathbf{x}_s{=}\boldsymbol{x}_s)$, and $q_{t|s}(\boldsymbol{x}_t|\boldsymbol{x}_s)$ when there is no ambiguity. We model the *forward diffusion* process independently for each node and edge, while the *backward denoising* process is modeled jointly for all nodes and edges. All vectors are column-wise vectors. Let $\langle \cdot, \cdot \rangle$ denote inner product.

3.1 Discrete Denoising Diffusion on Graphs

Denoising Diffusion is first developed by Sohl-Dickstein et al. [38] and later improved by Ho et al. [19]. It is further generalized to discrete-state case by Hoogeboom et al. [20] and Austin et al. [3]. Taking a graph G_0 as example, diffusion model defines a forward diffusion process to gradually inject noise to all nodes and edges independently until all reach a non-informative state G_T. Then, a denoising network is trained to reconstruct G_0 from the noisy sample G_t at each time step, by optimizing a Variational Lower Bound (VLB) for $\log p_\theta(\mathrm{G}_0)$. Specifically, the forward process is defined as a Markov chain with $q(\mathrm{G}_t|\mathrm{G}_{t-1}), \forall t \in [1,T]$, and the backward denoising process is parameterized with another Markov chain $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t), \forall t \in [1,T]$. Note that while the forward process is independently applied to all elements, the backward process is coupled together with conditional independence assumption. Formally,

$$q(\mathrm{G}_t|\mathrm{G}_{t-1}) = \prod_{i\in\mathcal{V}} q(\mathbf{v}_t^i|\mathbf{v}_{t-1}^i) \prod_{i,j\in\mathcal{V}} q(\mathbf{e}_t^{i,j}|\mathbf{e}_{t-1}^{i,j}), \quad p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t) = \prod_{i\in\mathcal{V}} p_\theta(\mathbf{v}_{t-1}^i|\mathrm{G}_t) \prod_{i,j\in\mathcal{V}} p_\theta(\mathbf{e}_{t-1}^{i,j}|\mathrm{G}_t)\,. \tag{1}$$

Then, the VLB of $\log p_\theta(\mathrm{G}_0)$ can be written (see Apdx.§A.1) as

$$\log p_\theta(\mathrm{G}_0) \geq \underbrace{\mathbb{E}_{q(\mathrm{G}_1|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_0|\mathrm{G}_1)\big]}_{-\mathcal{L}_1(\theta)} - \underbrace{D_{\mathrm{KL}}\big(q(\mathrm{G}_T|\mathrm{G}_0)||p_\theta(\mathrm{G}_T)\big)}_{\mathcal{L}_{\text{prior}}} - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\big[D_{\mathrm{KL}}\big(q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)||p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)\big)\big]}_{\mathcal{L}_t(\theta)} \tag{2}$$

where $\mathcal{L}_{\text{prior}} \approx 0$, since $p_\theta(\mathrm{G}_T) \approx q(\mathrm{G}_T|\mathrm{G}_0)$ is designed as a fixed noise distribution that is easy to sample from. To compute Eq. (2), we need to formalize the distributions (*i*) $q(\mathrm{G}_t|\mathrm{G}_0)$ and (*ii*)

https://doi.org/10.52202/079017-0229

$q(\mathrm{G}_{t-1}|\mathrm{G}_t, \mathrm{G}_0)$, as well as (*iii*) the parameterization of $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$. DiGress [44] applies D3PM [3] to define these three terms. Different from DiGress, we closely follow the approach in Zhao et al. [52] to define these three terms, as their formulation is simplified with improved memory usage and loss computation. For brevity, we refer readers to Appx. §A.2 for the details. Notice that while a neural network can directly be used to parameterize (*iii*) $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$, we follow [52] to parameterize $p_\theta(\mathrm{G}_0|\mathrm{G}_t)$ instead, and compute (*iii*) from $p_\theta(\mathrm{G}_0|\mathrm{G}_t)$.

With (*i* $\sim$ *iii*) known, one can compute the negative VLB loss in Eq. (2) exactly. In addition, at each time step t, the cross entropy (CE) loss between (*i*) $q(\mathrm{G}_t|\mathrm{G}_0)$ and $p_\theta(\mathrm{G}_0|\mathrm{G}_t)$ that quantifies reconstruction quality is often employed as an auxiliary loss, which is formulated as

$$\mathcal{L}_t^{CE}(\theta) = -\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\Big[\sum_{i\in\mathcal{V}} \log p_\theta(\mathbf{v}_0^i|\mathrm{G}_t) + \sum_{i,j\in\mathcal{V}} \log p_\theta(\mathbf{e}_0^{i,j}|\mathrm{G}_t)\Big] .$$

In fact, DiGress solely uses $\mathcal{L}_t^{CE}(\theta)$ to train their diffusion model. In this paper, we adopt a hybrid loss [3], that is $\mathcal{L}_t(\theta) + \lambda\mathcal{L}_t^{CE}(\theta)$ with $\lambda = 0.1$ at each time t, as we found it to help reduce overfitting. To generate a graph from $p_\theta(\mathrm{G}_0)$, a pure noise graph is first sampled from $p_\theta(\mathrm{G}_T)$ and gradually denoised using the learned $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$ from step T to 0.

A significant advantage of diffusion models is their ability to achieve exchangeable probability in combination with permutation equivariant networks under certain conditions [45]. DiGress is the first work that applied discrete denoising diffusion to graph generation, achieving significant improvement over previous continuous-state based diffusion. However, given the inherently high-dimensional nature of graphs and their complex internal dependencies, modeling the joint distribution of all nodes and edges directly presents significant challenges. DiGress requires thousands of denoising steps to accurately capture the original dependencies. Moreover, DiGress relies on many extra supplementary node and graph-level features, such as cycle counts and eigenvectors, to effectively break symmetries among structural equivalences to achieve high performance.

3.2 Autoregressive Graph Generation

Order is important for AR models. Unlike diffusion models that aim to capture the joint distribution directly, AR models decompose the joint probability into a product of simpler conditional probabilities based on an order. This makes AR models inherently suitable for ordinal data, where a natural order exists, such as in natural languages and images.

Order Sensitivity. Early works of graph generation contain many AR models like GraphRNN [48] and GRAN [28] based on non-deterministic heuristic node orders like BFS/DFS and k-core ordering. Despite being permutation sensitive, AR models achieve SOTA performance on small simulated structures like grid and lobster graphs. However, permutation invariance is necessary for estimating an accurate likelihood of a graph, and can benefit large-size datasets for better generalization.

Let π denote an ordering of nodes. To make AR order-*insensitive*, there are two directions: (1) Modeling the joint probability $p(\mathrm{G}, \pi)$ and then marginalizing π, (2) Finding a unique canonical order $\pi^*(\mathrm{G})$ for any graph G such that $p(\pi|\mathrm{G}) = 1$ if $\pi = \pi^*(\mathrm{G})$ and 0 otherwise. In direction (1), directly integrating out π is prohibitive as the number of permutations is factorial in the graph size. Several studies [27, 9, 28] have used subsets of either random or canonical orderings. This approach aims to simplify the process, but it results in approximated integrals with indeterminate errors. Moreover, it escalates computational expense due to the need for data augmentation involving these subsets of orderings. In direction (2), identifying a universal canonical order for all graphs is referred to as graph canonicalization. There exists no polynomial time solution for this task on general graphs, which is at least as challenging as the NP-intermediate Graph Isomorphism problem [2]. Goyal et al. [17] explored using minimum DFS code to construct canonical labels for a specific dataset with non-polynomial time complexity. However, the canonicalization is specific to each training dataset with the randomness derived from DFS. This results in a generalization issue, due to the canonical order being $\pi(\mathrm{G}|\mathrm{TrainSet})$ instead of $\pi(\mathrm{G})$.

The Existence of Partial Order. While finding a unique order for all nodes of a graph is NP-intermediate, we argue that finding a unique *partial* order, where certain nodes and edges are with the same rank, is easily achievable. For example, a trivial partial order is simply all nodes and edges having the same rank. Nevertheless, a graph is not the same as a set (a set is just a graph with empty $\mathcal{E}$), where all elements are essentially unordered with equivalent rank. That is because a non-empty graph contains edges between nodes, and these edges give different structural properties to nodes. Notice that some nodes or edges have the same structural property as they are structurally equivalent.

Algorithm 1 Structural Partial Order ϕ

Input: Graph G, maximum hops K_h.
Init: $G_0 = G$, $i = 0$, ϕ with $\phi(\boldsymbol{v}) = 0\ \forall \boldsymbol{v}$.
while G_i is not $\emptyset$ **do**
  Compute $w_{K_h}(\boldsymbol{v}), \forall \boldsymbol{v} \in \mathcal{V}(G_i)$, using Eq. (4).
  Find all nodes $\mathcal{L}$ with $w_{K_h} = \min_{\boldsymbol{v} \in \mathcal{V}(G)} w_{K_h}(\boldsymbol{v})$.
  Let $\phi(\boldsymbol{v}) = i\ \forall \boldsymbol{v} \in \mathcal{L}$.
  $G_{i+1} \leftarrow G_i[\mathcal{V}(G) \setminus \mathcal{L}]$; $i \leftarrow i + 1$.
end while
Output: $\phi \leftarrow i - \phi$

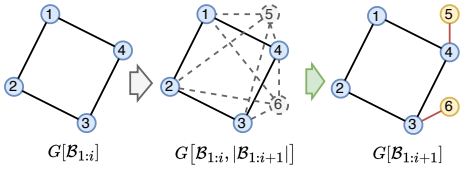


Figure 1: Example case where the equivariant graph transformation from $G[\mathcal{B}_{1:i}]$ to $G[\mathcal{B}_{1:i+1}]$ is impossible for *any* permutation-equivariant network due to structural equivalence of nodes. See Proposition 3.3.

We can view each structural property as a color, and rank all unique colors within the graph to define the partial order over nodes, which we call a *structural partial order*. The structural partial order defines a sequence of *blocks* such that all nodes within a block have the same rank (i.e. color).

Let $\phi : \mathcal{V} \to [1, ..., K_B]$ be the function that assigns rank to nodes based on their structural properties, where K_B denotes the maximum number of blocks. We use $G[\mathcal{S}]$ to denote the induced subgraph on the subset $\mathcal{S} \subseteq \mathcal{V}$. There are many ways to assign rank to structural colors, however we would like the resulting partial order to satisfy certain constraints. Most importantly, we want

$$\forall r \in [1, ..., K_B],\ G[\phi(\mathcal{V}) \leq r] \text{ is a connected graph.} \tag{3}$$

The connectivity requirement is to ensure a more accurate representation of real-world graph generation processes, where it is typical of real-world dynamic graphs to enlarge with newcoming nodes being connected at any time. Then, *one can sequentially remove all nodes with the lowest degree to maintain this connectivity and establish a partial order.* However, degree only reflects limited information of the first-hop neighbors, and many nodes share the same degree—leading to only a few distinct blocks, not significantly different from a trivial, single-block approach.

To ensure connectivity while reducing rank collision, we consider larger hops to define a weighted degree. Consider a maximum of K_h hops. For any node $\boldsymbol{v} \in \mathcal{V}$, the number of neighbors at each hop of $\boldsymbol{v}$ can be easily obtained as $[d_1(\boldsymbol{v}), ..., d_{K_h}(\boldsymbol{v})]$. We then define the weighted degree as

$$w_{K_h}(\boldsymbol{v}) = \sum_{k=1}^{K_h} d_k(\boldsymbol{v}) \times |\mathcal{V}|^{K_h - k} \tag{4}$$

Equation Eq. (4) may appear as an ad-hoc design, but it fundamentally serves as a hash that maps the vector $[d_1(\boldsymbol{v}), d_2(\boldsymbol{v}), ..., d_K(\boldsymbol{v})]$ to a unique scalar value. This mapping ensures a one-to-one correspondence between the vector and the scalar. Furthermore, it prioritizes lower-hop degrees over higher-hop degrees, akin to how e.g. the number "123" is represented as $1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$. Eq. (4) is efficient to compute and guarantees: 1) Nodes have the same rank if and only if they have the same number of neighbors up to K_h hops. 2) Lower-hop degrees are weighted more heavily. With w_{K_h} defined, we present our structural partial order in Algo. 1.

Proposition 3.1. *For any G, its structural partial order ϕ defined by Algo. 1 is permutation equivariant, such that $\phi(\boldsymbol{P} \star \mathrm{G}) = \boldsymbol{P} \star \phi(\mathrm{G})$ for any permutation operator $\boldsymbol{P}$.*

It is easy to prove Prop. 3.1. Algo. 1 shows that $\phi(i)$ for $\forall i \in \mathcal{V}$ is uniquely determined by node i's structural higher-order degree. As nodes' higher-order degree is permutation equivariant, ϕ is also permutation equivariant. Notice that ϕ is unique and deterministic for any graph.

Autoregressive Blockwise Generation. ϕ in Algo. 1 with output in range $[1, K_B]$ divides the nodes $\mathcal{V}(G)$ into K_B blocks $[\mathcal{B}_1, ..., \mathcal{B}_{K_B}]$ in order, where $\mathcal{B}_j = \{i \in \mathcal{V}(G) | \phi(i) = j\}$. Let $\mathcal{B}_{1:i} := \cup_{j=1}^{i} \mathcal{B}_j$ be the union of the first i blocks. PARD decomposes the joint probability of a graph G into

$$p_\theta(\mathrm{G}) = \prod_{i=1}^{K_B} p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \ \Big|\ \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) \tag{5}$$

where $\mathrm{G}[\mathcal{B}_{1:0}]$ is defined as the empty graph, and $\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]$ denotes the set of nodes and edges that are present in $\mathrm{G}[\mathcal{B}_{1:i}]$ but not in $\mathrm{G}[\mathcal{B}_{1:i-1}]$. All of them are represented in natural order of G. As each conditional probability only contains a subset of edges and nodes, and having access to all previous blocks, this conditional probability is significantly easier to model than the whole joint probability. Given the property Prop. 3.1 of $\mathcal{B}_i$, it is easy to verify that $p_\theta(\mathrm{G})$ is exchangeable with permutation-invariant probability for any G if and only if all conditional probabilities are exchangeable. See Appx. §A.6 for details of permutation-invariant probability. Note that while GRAN [28] also generates graphs block-by-block, all nodes within GRAN have different generation

https://doi.org/10.52202/079017-0229

ordering, even within the same block (it breaks symmetry for the problem identified later). Essentially, GRAN is an autoregressive method and, as such, suffers from all the disadvantages inherent to AR.

3.3 Impossibility of Equivariant Graph Transformation

In Eq. (5), we need to parameterize the conditional probability $p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\Big)$ to be permutation-invariant. This can be achieved by letting the conditional probability be

$$p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) \prod_{\mathbf{x}\in \mathrm{G}[\mathcal{B}_{1:i}]\setminus \mathrm{G}[\mathcal{B}_{1:i-1}]} p_\theta\Big(\mathbf{x}\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big) \tag{6}$$

where $\mathbf{x}$ is any node and edge in $\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]$, $\emptyset$ denotes an empty graph, hence $\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]$ depicts augmenting $\mathrm{G}[\mathcal{B}_{1:i-1}]$ with empty (or virtual) nodes and edges to the same size as $\mathrm{G}[\mathcal{B}_{1:i}]$. With the augmented graph, we can parameterize $p_\theta\big(\mathbf{x} \;\big|\; \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\big)$ for any node and edge $\mathbf{x}$ with a permutation equivariant network to achieve the required permutation invariance. For simplicity, let $\mathrm{G}\big[\mathcal{B}_{1:i-1}, |\mathcal{B}_i|\big] := \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]$.

The Flaw in Equivariant Modeling. Although the parameterization in Eq. (6) along with an equivariant network makes the conditional probability in Eq. (5) become permutation-invariant, we have found that the equivariant graph transformation $p_\theta(\mathbf{x} \mid \mathrm{G}\big[\mathcal{B}_{1:i-1}, |\mathcal{B}_i|\big])$ cannot be achieved in general for *any* permutation equivariant network, no matter how powerful it is (!) For definition of graph transformation, see Appx.§A.7. The underlying cause is the symmetry of structural equivalence, which is also a problem in link prediction [41, 49]. Formally, let $\boldsymbol{A}(G)$ be the adjacency matrix of G (ignoring labels) based on G's default node order, then an *automorphism* σ of G satisfies

$$\boldsymbol{A}(G) = \boldsymbol{A}(\sigma \star G) \tag{7}$$

where $\sigma \star G$ is a reordering of nodes based on the mapping σ. Then the automorphism group is

$$\mathrm{Aut}(G) = \{\sigma \in \mathbb{P}_{|\mathcal{V}|} \mid \boldsymbol{A}(G) = \boldsymbol{A}(\sigma \star G)\} \tag{8}$$

where $\mathbb{P}_n$ denotes all permutation mappings for size n. That is, $\mathrm{Aut}(G)$ contains all automorphisms of G. For a node i of G, the *orbit* that contains node i is defined as

$$o(i) = \{\sigma(i) \mid \forall \sigma \in \mathrm{Aut}(G)\}\ . \tag{9}$$

In words, the orbit $o(i)$ contains all nodes that are *structurally equivalent* to node i in G. Two edges (i, j) and (u, v) are structurally equivalent if $\exists \sigma \in \mathrm{Aut}(G)$, such that $\sigma(i) = u$ and $\sigma(j) = v$.

Theorem 3.2. *Any structurally equivalent nodes and edges will have identical representations in any equivariant network, regardless of its power or expressiveness.*

See proof in Apdx.§A.5. Theorem 3.2 indicates that no matter how powerful the equivariant network is, any structually equivalent elements have the same representation, which implies the following.

Proposition 3.3. *General graph transformation is not achievable with any equivariant model.*

Proof. To prove it, we only need to show there are many "bottleneck" cases where the transformation cannot be achieved. Fig. 1 shows a case where $G[\mathcal{B}_{1:}]$ is a 4-cycle, and the next target block contains two additional nodes, each with a single edge connecting to one of the nodes of $G[\mathcal{B}_{1:}]$. It is easy to see that nodes 1–4 are all structurally equivalent, and so are nodes $5, 6$ in the augmented case (middle). Hence, edges in $\{(5, i)|\forall i \in [1, 4]\}$ are structurally equivalent (also $\{(6, i)|\forall i \in [1, 4]\}$). Similarly, $\forall i \in [1, 4]$, edge $(5, i)$ and $(6, i)$ are structurally equivalent. Combining all cases, edges in $\{(j, i)|\forall i \in [1, 4], j \in \{5, 6\}\}$ are structurally equivalent. Theorem 3.2 states that all these edges would have the *same* prediction, hence making the target $G[\mathcal{B}_{1:i+1}]$ not achievable. □

3.4 PARD: Autoregressive Denoising Diffusion

The Magic of Annealing/Randomness. In Fig. 1 we showed that a graph with many automorphisms cannot be transformed to a target graph with fewer automorphisms. We hypothesize that *a graph with lower "energy" is hard to be transformed to a graph with higher "energy" with equivariant networks*. There exist some definitions and discussion of graph energy [18, 4] based on symmetry and eigen-information to measure graph complexity, where graphs with more symmetries have lower energy. The theoretical characterization of the conditions for successful graph transformation is a valuable direction, which we leave for future work to investigate.

Based on the above hypothesis, to achieve a successful transformation of a graph into a target graph, it is necessary to increase its energy. Since graphs with fewer symmetries exhibit higher energy levels, our approach involves adding random noise to nodes and edges. Our approach of elevating the energy level, followed by its reduction to attain desired target properties, mirrors the annealing process.

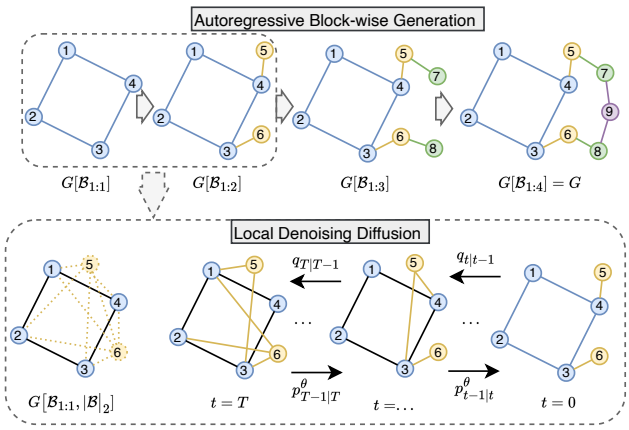


Figure 2: PARD integrates the autoregressive method with diffusion modeling. (top) PARD decomposes the joint probability into a series of block-wise enlargements, where each block's conditional distribution is captured with a shared discrete diffusion (bottom).

Diffusion. This further motivates us to use denoising diffusion to model $p_\theta(\mathbf{x} \mid \mathrm{G}[\mathcal{B}_{1:i-1}, |\mathcal{B}_i|])$: it naturally injects noise in the forward process, and its backward denoising process is the same as annealing. As we show below, this yields $p_\theta(G)$ in Eq. (5) to be permutation-invariant.

Theorem 3.4. PARD *is permutation-invariant such that* $p_\theta(\boldsymbol{P} \star \mathrm{G}) = p_\theta(\mathrm{G})\ \forall$ *permutator* $\boldsymbol{P}$.

The proof is given in Appx. §A.6. With all constituent parts presented, we summarize our proposed PARD, the first permutation-invariant autoregressive diffusion model that integrates AR with denoising diffusion. PARD relies on a unique, permutation equivariant structural partial order ϕ (Algo. 1) to decompose the joint graph probability to the product of simpler conditional probabilities, based on Eq. (5). Each block's conditional probability is modeled with the product of a conditional block size probability and a conditional block enlargement probability as in Eq. (6), where the latter for every block is a shared discrete denoising diffusion model as described in §3.1. Fig. 2 illustrates PARD's two parts : (top) block-wise AR and (bottom) local denoising diffusion at each AR step.

Notice that there are two tasks in Eq. (6); one for predicting the next block's size, and the other for predicting the next block's nodes and edges with diffusion. These two tasks can be trained together with a single network, although for better performance we use two different networks. For each block's diffusion model, we set the maximum time steps to 40 without much tuning.

Training and Inference. We provide the training and inference algorithms for PARD in Apdx.§A.8. Specifically, Algo. 2 is used to train next block's size prediction model; Algo. 3 is used to train the shared diffusion for block conditional probabilities; and Algo. 4 presents the generation steps.

4 Architecture Improvement

PARD is a general framework that can be combined with any equivariant network. Nevertheless, we would like an equivariant network with enough expressiveness to process symmetries inside the generated blocks for modeling the next block's conditional probability. While there are many expressive GNNs like subgraph GNNs [5, 50] and higher-order GNNs [51, 31], PPGN [30] is still a natural choice that models edge (2-tuple) representations directly with 3-WL expressivity and $O(n^3)$ complexity in graph size. However, PPGN's memory cost is relatively high for many datasets.

4.1 Efficient and Expressive Higher-order Transformer

To enhance the memory efficiency of PPGN while maintaining the expressiveness equivalent to the 3-Weisfeiler-Lehman (3-WL) test, we introduce a hybrid approach that integrates Graph Transformers with PPGN. Graph Transformers operate on nodes as the fundamental units of representation, offering better scalability and reduced memory consumption ($O(n^2)$) compared to PPGN. PPGN utilizes edges as their primary representation units and therefore incurs significantly higher memory requirements ($O(n^3)$). However, the expressiveness of Graph Transformers (without position encoding) is limited to the 1-WL test [8]. By combining these two models, we can drastically decrease the size of edge representations while allocating larger hidden sizes to nodes. This synergistic approach not only substantially lowers the memory footprint but also enhances overall performance, leveraging the

https://doi.org/10.52202/079017-0229

strengths of both architectures to achieve a balance between expressivity and efficiency. We provide the detailed design in Appx.§A.9. Note that we use GRIT [29] as the graph transformer block.

4.2 Parallel Training with Causal Transformer

As shown in Eq. (5), for a graph G, there are K_B conditional probabilities being modeled by a shared diffusion model using a θ-parameterized network f_θ. By default, these K_B number of inputs $\{\mathrm{G}[\mathcal{B}_{1:i-1}]\}_{i=1}^{K_B}$ are viewed as separate graphs and the representations during network passing $f_\theta(\mathrm{G}[\mathcal{B}_{1:i-1}])$ for different $i \in [1, K_B]$ are not shared. This leads to a scalability issue; in effect enlarging the dataset by roughly K_B times and resulting in K_B times longer training.

To minimize computational overhead, it is crucial to enable parallel training of all the K_B conditional probabilities, and allow these processes to share representations, through which we can pass the full graph G to the network f_θ only once and obtain all K_B conditional probabilities. This is also a key advantage of transformers over RNNs. Transformers (GPTs) can train all next-token predictions simultaneously with representation sharing through causal masking, whereas RNNs must train sequentially. However, the default causal masking of GPTs is not applicable to our architecture, as PARD contains both Transformer and PPGN where the PPGN's causal masking is not designed.

To ensure representation sharing without risking information leakage, we first assign a "block ID" to every node and edge within graph G. Specifically, for every node and edge in $G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}]$, we assign the ID equal to i. To prevent information leakage effectively, it is crucial that any node and edge labeled with ID i are restricted to communicate only with other nodes and edges whose ID is $\leq i$. Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$, and $\mathbf{x} \in \mathbb{R}^n$. There are mainly two non-elementwise operations in Transformer and PPGN that have the risk of leakage: the attention-vector product operation $\boldsymbol{A}\mathbf{x}$ of Transformer, and the matrix-matrix product operation $\boldsymbol{AB}$ of PPGN. (We ignore the feature dimension of $\boldsymbol{A}$ and $\mathbf{x}$ as it does not affect the information leakage.) Let $\boldsymbol{M} \in \{0,1\}^{n\times n}$ be a mask matrix, such that $\boldsymbol{M}_{i,j} = 1$ if block_ID(node i) $\geq$ block_ID(node j) else 0. One can verify that

$$(\boldsymbol{A} \odot \boldsymbol{M})\mathbf{x}, \quad \text{and} \quad (\boldsymbol{A} \odot \boldsymbol{M})\boldsymbol{B} + \boldsymbol{A}(\boldsymbol{B} \odot \boldsymbol{M}^\top) - (\boldsymbol{A} \odot \boldsymbol{M})(\boldsymbol{B} \odot \boldsymbol{M}^\top) \tag{10}$$

generalize $\boldsymbol{A}\mathbf{x}$ and $\boldsymbol{AB}$ respectively and safely bypass information leakage. We provide more details of parallel training and derivation of Eq. (10) in Appx.§A.10. We use these operations in our network and enable representation sharing, along with parallel training of all K_B blocks for denoising diffusion as well as next block size prediction. In practice, these offer more than $10\times$ speed-up, and the parallel training allows PARD to scale to large datasets like MOSES [33].

5 Experiments

We evaluate PARD on 8 diverse benchmark datasets with varying sizes and structural properties, including both molecular (§5.1) and non-molecular/generic (§5.2) graph generation. A summary of the datasets and details are in Appx. §A.11. Ablations and runtime measures are in Appx. §A.12.

5.1 Molecular Graph Generation

Datasets. We experiment with three different molecular datasets used across the graph generation literature: (1) QM9 [34] (2) ZINC250K [23], and (3) MOSES [33] that contains more than 1.9 million graphs. We use a 80%-20% train and test split, and among the train data we split additional 20% as validation. For QM9 and ZINC250K, we generate 10,000 molecules for stand-alone evaluation, and on MOSES we generate 25,000 molecules.

Baselines. The literature has not been consistent in evaluating molecule generation on well-adopted benchmark datasets and metrics. Among baselines, DiGress [44] stands out as the most competitive. We also compare to many other baselines shown in tables, such as GDSS [24] and GraphARM [25].

Metrics. The literature has adopted a number of different evaluation metrics that are not consistent across datasets. Most common ones include Validity (↑) , Uniqueness (↑) (frac. of valid molecules that are unique), and Novelty (↑) (frac. of valid molecules that are not included in the training set). For QM9, following earlier work [44], we report additional evaluations w.r.t. Atom Stability (↑) and Molecule Stability (↑), as defined by [22], whereas Novelty is not reported as explained in [44]. On ZINC250K and MOSES, we also measure the Fréchet ChemNet Distance (FCD) (↓) between the generated and the training samples, which is based on the embedding learned by ChemNet [27]. For MOSES, there are three additional measures: Filter (↑) score is the fraction of molecules passing the same filters as the test set, SNN (↑) evaluates nearest neighbor similarity using Tanimoto Distance, and Scaffold similarity (↑) analyzes the occurrence of Bemis-Murcko scaffolds [33].

Table 1: Generation quality on QM9 with explict hydrogens.

Model	Valid. ↑	Uni. ↑	Atom.↑	Mol. ↑
Dataset (optimal)	97.8	100	98.5	87.0
ConGress	86.7	**98.4**	97.2	69.5
DiGress (uniform)	89.8	97.8	97.3	70.5
DiGress (marginal)	92.3	97.9	97.3	66.8
DiGress (marg. + *feat.*)	95.4	97.6	98.1	79.8
PARD (*no feat.*)	**97.5**	95.8	**98.4**	**86.1**

Table 2: Generation quality on ZINC250K.

Model	Validity ↑	FCD ↓	Uni. ↑	Model Size
EDP-GNN	82.97	16.74	99.79	0.09M
GraphEBM	5.29	35.47	98.79	-
SPECTRE	90.20	18.44	67.05	-
GDSS	**97.01**	14.66	99.64	0.37M
GraphArm	88.23	16.26	99.46	-
DiGress	91.02	23.06	81.23	18.43M
SwinGNN-L	90.68	1.99	99.73	35.91M
PARD	95.23	**1.98**	**99.99**	4.1M

Results. Table 1 shows generation evaluation results on QM9, where the baseline results are sourced from [44]. PARD outperforms DiGress and variants that do *not* use any auxiliary features, with slightly lower Uniqueness. What is notable is that PARD, without using any extra features, achieves a similar performance gap against DiGress that uses specialized extra features. Table 2 shows PARD's performance on ZINC250K, with baseline results carried over from [25] and [46]. PARD achieves the best Uniqueness, stands out in FCD alongside SwinGNN [46], and is the runner-up w.r.t. Validity.

Table 3: Generation quality on MOSES. The top three methods use hard-coded rules (not highlight).

Model	Val. ↑	Uni. ↑	Novel. ↑	Filters ↑	FCD ↓	SNN ↑	Scaf. ↑
VAE	97.7	99.8	69.5	99.7	0.57	0.58	5.9
JT-VAE	100	100	99.9	97.8	1.00	0.53	10.0
GraphINVENT	96.4	99.8	-	95.0	1.22	0.54	12.7
ConGress	83.4	99.9	**96.4**	94.8	1.48	0.50	**16.4**
DiGress	85.7	**100**	95.0	97.1	1.19	0.52	14.8
PARD	**86.8**	**100**	78.2	**99.0**	**1.00**	**0.56**	2.2

Finally, Table 3 shows generation quality on the largest dataset MOSES. We mainly compare with DiGress and its variants, which has been the only general-purpose generative model in the literature that is not based on molecular fragments or SMILES strings. Baselines are sourced from [44]. While other specialized models, excluding PARD and DiGress, have hard-coded rules to ensure high Validity, PARD outperforms those on several other metrics including FCD and SNN, and achives competitive performance on others. Again, it is notable here that PARD, *without* relying on any auxiliary features, achieves similarly competitive results as with DiGress which utilizes extra features.

5.2 Generic Graph Generation

Table 4: Generation quality on generic graphs. All metrics are based on generated-to-test set MMD distances, the lower the better. Top performance is in **bold**, and Runner-up is underlined.

	COMMUNITY-SMALL			CAVEMAN			CORA			BREAST			GRID		
Model	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit
GraphRNN	0.080	0.120	0.040	0.371	1.035	0.033	1.689	0.608	0.308	0.103	0.138	0.005	0.064	0.043	0.021
GRAN	0.060	0.110	0.050	0.043	0.130	0.018	0.125	0.272	0.127	0.073	0.413	0.010	-	-	-
EDP-GNN	0.053	0.144	0.026	0.032	0.168	0.030	0.093	0.269	0.062	0.131	0.038	0.019	0.455	0.238	0.328
GDSS	0.045	0.086	0.007	0.019	0.048	0.006	0.160	0.376	0.187	0.113	**0.020**	0.003	0.111	0.005	0.070
GraphArm	0.034	0.082	**0.004**	0.039	**0.028**	0.018	0.273	0.138	0.105	**0.036**	0.041	0.002	-	-	-
DiGress	0.047	**0.041**	0.026	0.019	0.040	0.003	0.044	0.042	0.223	0.152	0.024	0.008	-	-	-
PARD	**0.023**	0.071	0.012	**0.002**	0.047	**0.00003**	**0.0003**	**0.003**	**0.0097**	0.044	0.024	**0.0003**	**0.028**	**0.002**	**0.029**

Datasets. We use five generic graph datasets with various structure and semantic: (1) COMMUNITY-SMALL [48], (2) CAVEMAN [47], (3) CORA [35], (4) BREAST [15], and (5) GRID [48]. We split each dataset into 80%-20% train-test, and randomly sample 20% of training graphs for validation. We generate the same number of samples as the test set. Notice that GRID contains graphs with 100∼400 nodes, which is relatively large for diffusion models. There are lots of symmetries inside, hence it is difficult to capture all dependencies with permutation-equivariant models.

Baselines. We mainly compare against the latest general-purpose GraphArm [25], which reported DiGress [44] and GDSS [24] as top two most competitive, along with several other baselines.

Metrics. We follow [48] to measure generation quality using the maximum mean discrepancy (MMD) as a distribution distance between the generated graphs and the test graphs (↓), as pertain to distributions of (i) Degree, (ii) Clustering coefficient, and (iii) occurrence count of all Orbits.

https://doi.org/10.52202/079017-0229

Results. Table 4 provides the generation results of PARD against the baselines as sourced from [25]. PARD shows outstanding performance achieving SOTA or close runner-up results, while none of the baselines shows as consistent performance across datasets and metrics.

5.3 Ablation Study

Q1: do we really need AR in diffusion, given diffusion is already permutation-invariant?

In DiGress [44], we observed that pure diffusion, while being permutation-invariant, requires (1) many sampling/denoising steps to break symmetries and (2) additional features like eigenvectors to further break symmetry. This shows that directly capturing the FULL joint distribution and solving the transformation difficulty (in Sec. 3.3) via diffusion is challenging. Additionally, AR methods still dominate LLMs, indicating their potential to benefit diffusion models. To quantitatively verify our analysis, we perform an ablation study on the maximum number of hops, K_h, which controls the extent of autoregression (AR). When $K_h = 0$, all nodes have a fixed degree of 1, resulting in a single block per graph, equivalent to full diffusion without AR. As K_h increases, more blocks are generated with smaller average block sizes, indicating a greater number of AR steps.

Table 5: Ablation study on QM9 with varying maximum hops while keeping the total diffusion steps fixed (first two parts). The last part examines the effect of increasing steps for the no AR case.

Setting	**No AR**	**With AR**			**No AR, ↑ steps**	
Total diffusion steps	140	140			280	490
Maximum hops	0	1	2	3	0	0
Average number of blocks	1	4.3	5.6	7.75	1	1
Diffusion steps per block	140	32	25	20	280	490
Validity	93.8	**97.1**	96.7	97.0	94.3	95.2
Uniqueness	**96.9**	96.5	96.2	96.1	96.5	96.9
Mol stability	76.4	86.1	85.4	**86.3**	79.3	79.2
Atom Stability	97.7	98.3	98.3	**98.4**	97.9	98.0

Table 5 shows the result of the controlled experiment with 140 total diffusion steps across all trials, using the same model architecture, diffusion algorithm, and training settings. The significant improvement from $K_h = 0$ to $K_h = 1$ confirms that our enhancement stems from breaking the full joint distribution into several conditional distributions. Furthermore, adding more diffusion steps to full diffusion approach does not close the gap to AR enhanced diffusion, indicates the necessary of combining AR and diffusion.

Q2: how does model architecture affect performance?

Table 6: Results for QM9 dataset with different model architectures with $K_h = 3$ and 140 total steps.

Backbone	**Transformer**	**PPGN**	**PPGNTransformer**
Validity	26.3	96.6	97.1
Uniqueness	94.5	96.3	96.0
Mol Stability	17.6	84.67	86.2
Atom Stability	81.4	98.2	98.4

Table 6 this indicates that PPGN component is essential for diffusion with autoregression. The design of combining PPGN and transformer in Sec. 4 further addresses the efficiency of PPGN.

Other ablations: other ablations and runtime measures are in Appx. §A.12

6 Conclusion

We presented PARD, the *first* permutation-invariant autoregressive diffusion model for graph generation. PARD decomposes the joint probability of a graph autoregressively into the product of several block conditional probabilities, by relying on a unique and permutation equivariant structural partial order. All conditional probabilities are then modeled with a shared discrete diffusion. PARD can be trained in parallel on all blocks, and efficiently scales to millions of graphs. PARD achieves SOTA performance on molecular and non-molecular datasets without using any extra features. Notably, we expect PARD to serve as a cornerstone toward generative foundation modeling for graphs.

References

[1] Namrata Anand and Possu Huang. Generative modeling for protein structures. *Advances in neural information processing systems*, 31, 2018.

[2] Vikraman Arvind, Bireswar Das, and Johannes Köbler. The space complexity of k-tree isomorphism. In *Algorithms and Computation: 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007. Proceedings 18*, pages 822–833. Springer, 2007.

[3] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[4] R Balakrishnan. The energy of a graph. *Linear Algebra and its Applications*, 387:287–295, 2004.

[5] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.

[6] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[7] Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. Graph generators: State of the art and open challenges. *ACM computing surveys (CSUR)*, 53(2):1–30, 2020.

[8] Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer. *International Conference on Machine Learning*, 2023.

[9] Xiaohui Chen, Xu Han, Jiajing Hu, Francisco Ruiz, and Liping Liu. Order matters: Probabilistic modeling of node sequence for graph generation. In *International Conference on Machine Learning*, pages 1630–1639. PMLR, 2021.

[10] Suparna De, Maria Bermudez-Edo, Honghui Xu, and Zhipeng Cai. Deep generative models in the industrial internet of things: a survey. *IEEE Transactions on Industrial Informatics*, 18(9): 5728–5737, 2022.

[11] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[12] William A Falcon. Pytorch lightning. *GitHub*, 3, 2019.

[13] Wenqi Fan, Chengyi Liu, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. Generative diffusion models on graphs: Methods and applications. *arXiv preprint arXiv:2302.02591*, 2023.

[14] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[15] Laura Gonzalez-Malerva, Jaehong Park, Lihua Zou, Yanhui Hu, Zahra Moradpour, Joseph Pearlberg, Jacqueline Sawyer, Hallam Stevens, Ed Harlow, and Joshua LaBaer. High-throughput ectopic expression screen for tamoxifen resistance identifies an atypical kinase that blocks autophagy. *Proceedings of the National Academy of Sciences*, 108(5):2058–2063, 2011.

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

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

[18] Ivan Gutman, Xueliang Li, and Jianbin Zhang. Graph energy. *Analysis of Complex Networks: From Biology to Linguistics*, pages 145–174, 2009.

 https://doi.org/10.52202/079017-0229

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[20] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021.

[21] Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2022.

[22] Emiel Hoogeboom, Víctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3D. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 8867–8887. PMLR, 17–23 Jul 2022.

[23] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52 (7):1757–1768, 2012.

[24] 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*, pages 10362–10383. PMLR, 2022.

[25] Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *International Conference on Machine Learning*, pages 17391–17408. PMLR, 2023.

[26] Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, 10:1–24, 2018.

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

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

[29] Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, K. Dokania, Mark Coates, Philip H.S. Torr, and Ser-Nam Lim. Graph Inductive Biases in Transformers without Message Passing. In *Proc. Int. Conf. Mach. Learn.*, 2023.

[30] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

[31] Christopher Morris, Gaurav Rattan, Sandra Kiefer, and Siamak Ravanbakhsh. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *International Conference on Machine Learning*, pages 16017–16042. PMLR, 2022.

[32] 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*, pages 4474–4484. PMLR, 2020.

[33] Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, Artur Kadurin, Simon Johansson, Hongming Chen, Sergey Nikolenko, Alan Aspuru-Guzik, and Alex Zhavoronkov. Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *Frontiers in Pharmacology*, 2020.

[34] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[35] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[36] Chence Shi*, Minkai Xu*, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

[37] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

[38] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and sunrya Ganguli. Deep unsunpervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[39] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[40] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[41] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*, 2020.

[42] Xiaochu Tong, Xiaohong Liu, Xiaoqin Tan, Xutong Li, Jiaxin Jiang, Zhaoping Xiong, Tingyang Xu, Hualiang Jiang, Nan Qiao, and Mingyue Zheng. Generative models for de novo drug design. *Journal of Medicinal Chemistry*, 64(19):14011–14027, 2021.

[43] Jeanne Trinquier, Guido Uguzzoni, Andrea Pagnani, Francesco Zamponi, and Martin Weigt. Efficient generative modeling of protein sequences using simple autoregressive models. *Nature communications*, 12(1):5800, 2021.

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

[45] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2022.

[46] Qi Yan, Zhengyang Liang, Yang Song, Renjie Liao, and Lele Wang. Swingnn: Rethinking permutation invariance in diffusion models for graph generation. *arXiv preprint arXiv:2307.01646*, 2023.

[47] Jiaxuan You. Caveman Dataset. https://github.com/JiaxuanYou/graph-generation/blob/master/create_graphs.py, 2018. [Online; accessed 31-Jan-2024].

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

[49] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

[50] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022.

[51] Lingxiao Zhao, Neil Shah, and Leman Akoglu. A practical, progressively-expressive gnn. *Advances in Neural Information Processing Systems*, 35:34106–34120, 2022.

[52] Lingxiao Zhao, Xueying Ding, Lijun Yu, and Leman Akoglu. Improving and unifying discrete&continuous-time discrete denoising diffusion. *arXiv preprint arXiv:2402.03701*, 2024.

 https://doi.org/10.52202/079017-0229

A Appendix

A.1 Variational Lower Bound Derivation

$$
\begin{aligned}
&\log \int q(\mathrm{G}_{1:T}|\mathrm{G}_0)\frac{p_\theta(\mathrm{G}_{0:T})}{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}d\mathrm{G}_{1:T} \geq \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_{0:T}) - \log q(\mathrm{G}_{1:T}|\mathrm{G}_0)\big] \\
&= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_{0:T}) + \sum_{t=1}^{T}\log\frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_t|\mathrm{G}_{t-1})}\big] \\
&= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathbf{x}_0)}\big[\log p_\theta(\mathrm{G}_T) + \log\frac{p_\theta(\mathrm{G}_0|\mathrm{G}_1)}{q(\mathrm{G}_1|\mathrm{G}_0)} + \sum_{t=2}^{T}\log\frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_t|\mathrm{G}_{t-1},\mathrm{G}_0)}\big] \\
&= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_T) + \log\frac{p_\theta(\mathrm{G}_0|\mathrm{G}_1)}{q(\mathrm{G}_1|\mathrm{G}_0)} + \sum_{t=2}^{T}\log\left(\frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)}\cdot\frac{q(\mathrm{G}_{t-1}|\mathrm{G}_0)}{q(\mathrm{G}_t|\mathrm{G}_0)}\right)\big] \\
&= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_T) + \log\frac{p_\theta(\mathrm{G}_0|\mathrm{G}_1)}{q(\mathrm{G}_1|\mathrm{G}_0)} + \log\frac{q(\mathrm{G}_1|\mathrm{G}_0)}{q(\mathrm{G}_T|\mathrm{G}_0)} + \sum_{t=2}^{T}\log\frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)}\big] \\
&= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_0|\mathrm{G}_1) + \log\frac{p_\theta(\mathrm{G}_T)}{q(\mathrm{G}_T|\mathrm{G}_0)} - \sum_{t=2}^{T}\log\frac{q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)}{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}\big] \\
&= \underbrace{\mathbb{E}_{q(\mathrm{G}_1|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_0|\mathrm{G}_1)\big]}_{-\mathcal{L}_1(\theta)} - \sum_{t=2}^{T}\underbrace{\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\big[D_{\mathrm{KL}}\big(q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)||p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)\big)\big]}_{\mathcal{L}_t(\theta)} - \text{const.}
\end{aligned}
\tag{11}
$$

Using Eq. (1), the first term can simplified as

$$
\mathbb{E}_{q(\mathrm{G}_1|\mathrm{G}_0)}[\sum_i \log p_\theta(\mathbf{v}_0^i|\mathrm{G}_1) + \sum_{i,j}\log p_\theta(\mathbf{e}_0^{i,j}|\mathrm{G}_1)]\ , \tag{12}
$$

and similarly, the t-th step loss $\mathcal{L}_t(\theta)$ is

$$
\begin{aligned}
\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\big[&\sum_i KL\big(q(\mathbf{v}_{t-1}^i|\mathbf{v}_t^i,\mathbf{v}_0^i)\ ||\ p_\theta(\mathbf{v}_{t-1}^i|\mathrm{G}_t)+ \\
&\sum_{i,j} KL\big(q(\mathbf{e}_{t-1}^{i,j}|\mathbf{e}_t^{i,j},\mathbf{e}_0^{i,j})\ ||\ p_\theta(\mathbf{e}_{t-1}^{i,j}|\mathrm{G}_t)\big]
\end{aligned}
\tag{13}
$$

A.2 Details of Discrete Diffusion Used in Paper

We closely follow the approach in Zhao et al. [52] to define forward and backward process in discrete diffusion, and get the formulation of three important distributions, (i) $q(\mathrm{G}_t|\mathrm{G}_0)$ (ii) $q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)$, and ($iii$) $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$, needed for computing negative VLB based loss. In here, we explain the detailed constructions. Notice that we only use the most basic discrete-time discrete diffusion formulation in Zhao et al. [52], mainly for improved memory efficiency over other old approaches like D3PM [3]. Other enhanced techniques in [52] like approximated loss and continuous-time formulation are not used, to keep the discrete diffusion part simple.

DiGress [44] applies D3PM [3] to define these three terms, our approach is similar. Since all elements in the forward process are independent as shown in Eq. (1), one can verify that the two terms $q(\mathrm{G}_t|\mathrm{G}_0)$ and $q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)$ are in the form of a product of independent distributions on each element. For simplicity, we introduce the formulation for a single element $\mathbf{x}$, with $\mathbf{x}$ being $\mathbf{v}^i$ or $\mathbf{e}^{i,j}$. We assume each discrete random variable $\mathbf{x}_t$ has a categorical distribution, i.e. $\mathbf{x}_t \sim \mathrm{Cat}(\mathbf{x}_t;\boldsymbol{p})$ with $\boldsymbol{p}\in[0,1]^K$ and $\mathbf{1}^\top\boldsymbol{p}=1$. One can verify that $p(\mathbf{x}_t=\boldsymbol{x}_t)=\boldsymbol{x}_t^\top\boldsymbol{p}$, or simply $p(\mathbf{x}_t)=\boldsymbol{x}_t^\top\boldsymbol{p}$. As shown in Hoogeboom et al. [20], Austin et al. [3], the forward process with discrete variables $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ can be represented as a transition matrix $Q_t\in[0,1]^{K\times K}$ such that $[Q_t]_{ij}=q(\mathbf{x}_t=\boldsymbol{e}_j|\mathbf{x}_{t-1}=\boldsymbol{e}_i)$. Then,

$$
q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathrm{Cat}(\mathbf{x}_t;Q_t^\top\boldsymbol{x}_{t-1})\ . \tag{14}
$$

Given transition matrices $Q_1, ..., Q_T$, the forward conditional marginal distribution is

$$(i)\ q(\mathbf{x}_t|\mathbf{x}_0) = \text{Cat}(\mathbf{x}_t; \overline{Q}_t^\top \boldsymbol{x}_0), \text{with } \overline{Q}_t = Q_1 ... Q_t\ , \tag{15}$$

and the $(t-1)$-step posterior distribution can be written as

$$(ii)\ \ q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \text{Cat}(\mathbf{x}_{t-1}; \frac{Q_t \boldsymbol{x}_t \odot \overline{Q}_{t-1}^\top \boldsymbol{x}_0}{\boldsymbol{x}_t^\top \overline{Q}_t^\top \boldsymbol{x}_0})\ . \tag{16}$$

See Apdx.§A.3 for the derivation. We have the option to specify node- or edge-specific quantities, $Q_t^{v,i}$ and $Q_t^{e,i,j}$, respectively, or allow all nodes and edges to share a common Q_t^v and Q_t^e. Leveraging Eq. (15) and Eq. (16), we can precisely determine $q(\mathbf{v}_t^i|\mathbf{v}_0^i)$ and $q(\mathbf{v}_{t-1}^i|\mathbf{v}_t^i, \mathbf{v}_0^i)$ for every node, and a similar approach can be applied for the edges. To ensure simplicity and a non-informative $q(\mathrm{G}_T|\mathrm{G}_0)$ (see Apdx.§A.4), we choose

$$Q_t = \alpha_t I + (1 - \alpha_t)\mathbf{1}\boldsymbol{m}^\top \tag{17}$$

for all nodes and edges, where $\alpha_t \in [0, 1]$, and $\boldsymbol{m}$ is a uniform distribution ($\mathbf{1}/K_v$ for nodes and $\mathbf{1}/K_e$ for edges). Note that DiGress [44] chooses $\boldsymbol{m}$ as the marginal distribution of nodes and edges. As $p(\mathbf{x}_{t-1}|\mathbf{x}_t) = \sum_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)p(\mathbf{x}_0|\mathbf{x}_t)$, the parameterization of $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$ can use the relationship, with

$$(iii)\quad p_\theta(\mathbf{x}_{t-1}|\mathrm{G}_t) = \sum_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)p_\theta(\mathbf{x}_0|\mathrm{G}_t) \tag{18}$$

where $\mathbf{x}$ can be any $\mathbf{v}^i$ or $\mathbf{e}^{i,j}$. With Eq. (18), we can parameterize $p_\theta(\mathbf{x}_0|\mathrm{G})$ directly with a neural network, and compute the negative VLB loss in Eq. (2) exactly, using Eq.s (15), (16) and (18).

A.3 Derivation of $q(x_{t-1}|x_t, x_0)$

First, define $\overline{Q}_{t|s} = Q_{s+1} ... Q_t$. Note that $\overline{Q}_{t|0} = \overline{Q}_t$ and $\overline{Q}_{t|t-1} = Q_t$. Accordingly, we can derive the following two equalities.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \text{Cat}(\mathbf{x}_t; \overline{Q}_t^\top \mathbf{x}_{t-1}) \tag{19}$$

$$\begin{aligned} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \frac{\text{Cat}(\mathbf{x}_t; Q_t^\top \mathbf{x}_{t-1})\text{Cat}(\mathbf{x}_{t-1}; \overline{Q}_{t-1}^\top \mathbf{x}_0)}{\text{Cat}(\mathbf{x}_t; \overline{Q}_t^\top \mathbf{x}_0)} \\ &= \frac{\mathbf{x}_{t-1}^\top Q_t \mathbf{x}_t \cdot \mathbf{x}_{t-1}^\top \overline{Q}_{t-1}^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0} = \mathbf{x}_{t-1}^\top \frac{Q_t \mathbf{x}_t \odot \overline{Q}_{t-1}^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0} = \text{Cat}(\mathbf{x}_{t-1}; \frac{Q_t \mathbf{x}_t \odot \overline{Q}_{t-1}^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0}) \end{aligned} \tag{20}$$

A.4 Simplification of Transition Matrix

For *any* transition matrices $Q_1, ..., Q_T$, which however should be chosen such that every row of $\overline{Q}_t = \overline{Q}_{t|0}$ converge to the same known stationary distribution when t becomes large (i.e. at T), let the known stationary distribution be $\mathbf{m}_0 \sim \text{Cat}(\mathbf{m}_0; \boldsymbol{m})$. Then, the constraint can be stated as

$$\lim_{t \to T} \overline{Q}_t = \mathbf{1}\boldsymbol{m}^\top\ . \tag{21}$$

To achieve the desired convergence on nominal data (which is typical data type in edges and nodes of graphs), while keeping the flexibility of choosing any categorical stationary distribution $\mathbf{m}_0 \sim \text{Cat}(\mathbf{m}_0; \boldsymbol{m})$, we define Q_t as

$$Q_t = \alpha_t I + (1 - \alpha_t)\mathbf{1}\boldsymbol{m}^\top\ , \tag{22}$$

where $\alpha_t \in [0, 1]$. This results in the accumulated transition matrix $\overline{Q}_{t|s}$ being equal to

$$\overline{Q}_{t|s} = \overline{\alpha}_{t|s} I + (1 - \overline{\alpha}_{t|s})\mathbf{1}\boldsymbol{m}^\top\ \forall t > s\ , \tag{23}$$

where $\overline{\alpha}_{t|s} = \prod_{i=s+1}^{t} \alpha_i$. Note that $\overline{\alpha}_t = \overline{\alpha}_{t|0} = \overline{\alpha}_{t|s}\overline{\alpha}_s$. We achieve Eq. (21) by picking α_t such that $\lim_{t \to T} \overline{\alpha}_t = 0$.

https://doi.org/10.52202/079017-0229

A.5 Proof of Theorem 3.2

We prove this for node case. For structurally equivalent edges, the analysis is the same. Assume node i and node j are structually equivalent, then we can find an automorphism $\sigma \in \mathrm{Aut}(G)$ such that $\sigma(i) = j$. For any permutation $\boldsymbol{P} \in \mathbb{P}_{|G|}$ and an equivariant network f, we have

$$f(\boldsymbol{P} \star \mathrm{G}) = \boldsymbol{P} \star f(\mathrm{G}) \tag{24}$$

Replace $\boldsymbol{P}$ with σ, and using the fact that $\sigma \star \mathrm{G} = \mathrm{G}$. We can get

$$f(\mathrm{G}) = f(\sigma \star \mathrm{G}) = \sigma \star f(\mathrm{G}) \tag{25}$$

Hence, we get $f(\mathrm{G})_i = f(\mathrm{G})_j$, that is two nodes i and j have the same representation.

A.6 Proof of Exchangeable Probability

To prove that $p_\theta(\mathrm{G})$ is an exchangeable probability, we need to show that for any permutation operator $\boldsymbol{P}$ with $\boldsymbol{P} \star \mathrm{G} = (\boldsymbol{P}\mathrm{V}, \boldsymbol{P}\mathrm{E}\boldsymbol{P}^\top)$, the probability satisfies $p_\theta(\boldsymbol{P} \star \mathrm{G}) = p_\theta(\mathrm{G})$. While $\boldsymbol{P}$ is defined as a group acting on all nodes ($|V|$) in the graph G, this operator can also be applied directly to sub-components of G, in such a manner that it permutes only the elements within these components. We use $\boldsymbol{P}$ to these sub-components directly in the following proof.

Recall that PARD define $p_\theta(\mathrm{G})$ as

$$p_\theta(\mathrm{G}) = \prod_{i=1}^{K_B} p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\Big). \tag{26}$$

Observe that for a graph G, $\mathcal{B}_i$ represents a subset containing certain nodes from G. When G is represented in its default order G, the subset $\mathcal{B}_i$ can be represented as a binary vector mask $\in \{0,1\}^{|\mathcal{V}|}$, where a value of 1 at the j-th position indicates that the j-th node in G is included in the subset. Then $\mathrm{G}[\mathcal{B}_i]$ can be view as indexing nodes and edges from G using mask $\mathcal{B}_i$. As indexing operation is permutation equivariant, we have $\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}]) = (\boldsymbol{P} \star \mathrm{G})[\boldsymbol{P} \star \mathcal{B}_{1:i}]$. What is more, $\mathcal{B}_i$ is actually a function of G and can be represented as $\mathcal{B}_{1:i}(\mathrm{G})$. This function is actually permutation equivariant, such that $\mathcal{B}_i(\boldsymbol{P} \star \mathrm{G}) = \boldsymbol{P} \star \mathcal{B}_i(\mathrm{G})$, as $\mathcal{B}_i(\mathrm{G})$ is determined by algorithm 1 solely based on structural degree information (up to K hops) of each node, where the structural feature is permutation equivariant.

Then we have

$$\begin{aligned}
&p_\theta(\boldsymbol{P} \star \mathrm{G}) \\
&= \prod_{i=1}^{K_B} p_\theta\Big((\boldsymbol{P} \star \mathrm{G})[\mathcal{B}_{1:i}(\boldsymbol{P} \star \mathrm{G})] \setminus (\boldsymbol{P} \star \mathrm{G})[\mathcal{B}_{1:i-1}(\boldsymbol{P} \star \mathrm{G})] \;\Big|\; (\boldsymbol{P} \star \mathrm{G})[\mathcal{B}_{1:i-1}(\boldsymbol{P} \star \mathrm{G})]\Big) \\
&= \prod_{i=1}^{K_B} p_\theta\Big((\boldsymbol{P} \star \mathrm{G})[\boldsymbol{P} \star \mathcal{B}_{1:i}] \setminus (\boldsymbol{P} \star \mathrm{G})[\boldsymbol{P} \star \mathcal{B}_{1:i-1}] \;\Big|\; (\boldsymbol{P} \star \mathrm{G})[\boldsymbol{P} \star \mathcal{B}_{1:i-1}]\Big) \quad \text{(Prop. 3.1)} \\
&= \prod_{i=1}^{K_B} p_\theta\Big(\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}]) \setminus \boldsymbol{P} \star \mathrm{G}[\mathcal{B}_{1:i-1}] \;\Big|\; \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}])\Big) \quad \text{(Indexing's equivariance)} \\
&= \prod_{i=1}^{K_B} p_\theta\Big(\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]) \;\Big|\; \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}])\Big) \\
&= \prod_{i=1}^{K_B} p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \boldsymbol{P} \star \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) p_\theta\Big(\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]) \Big| \; \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big)
\end{aligned} \tag{27}$$

Notice that we are using a permutation invariant function to approximate $p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \boldsymbol{P} \star \mathrm{G}[\mathcal{B}_{1:i-1}]\Big)$, hence we have

$$p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \boldsymbol{P} \star \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) = p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\Big). \tag{28}$$

For the second part, notice that

$$p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \Big| \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big) = \int p_\theta\Big(\mathrm{H}[\mathcal{B}_{1:i-1}] \cup \big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]\big) \Big| \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big) d\mathrm{H}[\mathcal{B}_{1:i-1}] \tag{29}$$

Then we have

$$\begin{aligned}
&p_\theta\Big(\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]) \Big| \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) \\
&= \int p_\theta\Big(\mathrm{H}[\mathcal{B}_{1:i-1}] \cup \boldsymbol{P} \star \big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]\big) \Big| \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) d\mathrm{H}[\mathcal{B}_{1:i-1}] \\
&= \int p_\theta\Big(\boldsymbol{P} \star \mathrm{H}[\mathcal{B}_{1:i-1}] \cup \boldsymbol{P} \star \big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]\big) \Big| \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) d\boldsymbol{P} \star \mathrm{H}[\mathcal{B}_{1:i-1}] \\
&= \int p_\theta\Big(\boldsymbol{P} \star \big(\mathrm{H}[\mathcal{B}_{1:i-1}] \cup \big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]\big)\big) \Big| \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) d\mathrm{H}[\mathcal{B}_{1:i-1}]
\end{aligned} \tag{30}$$

Now notice that the probability $p_\theta\Big(\boldsymbol{P}\star\big(\mathrm{H}[\mathcal{B}_{1:i-1}]\cup\big(\mathrm{G}[\mathcal{B}_{1:i}]\setminus\mathrm{G}[\mathcal{B}_{1:i-1}]\big)\big) \Big| \boldsymbol{P}\star(\mathrm{G}[\mathcal{B}_{1:i-1}]\cup\emptyset[\mathcal{B}_{1:i}])\Big)$ is actually a conditional distribution from one graph to another graph (with same size), and we can simplify it as $p_\theta(\mathrm{H}|\mathrm{G})$. As this part is modeled by diffusion model, we prove that this function is permutation equivariant. That is, for any permutation $\boldsymbol{P}$,

$$p_\theta(\mathrm{H}|\mathrm{G}) = p_\theta(\boldsymbol{P} \star \mathrm{H}|\boldsymbol{P} \star \mathrm{G}) \tag{31}$$

Proof. Our diffusion process is using an equivariant model through a markov chain that from G to H_T, and then from H_T to H_{T-1}, ..., H_t to H_{t-1}, until H_1 to $\mathrm{H}_0 = \mathrm{H}$. Then we have

$$p_\theta(\mathrm{H}|\mathrm{G}) = p_\theta(\mathrm{H}_0|\mathrm{G}) = \int p_\theta(\mathrm{H}_T|\mathrm{G}) \prod_{t=1}^{T} p_\theta(\mathrm{H}_t|\mathrm{H}_{t-1}) d\mathrm{H}_{1:T} \tag{32}$$

Because we are using an shared permutation equivariant model to model all $p_\theta(\mathrm{H}_t|\mathrm{H}_{t-1})$, we have $p_\theta(\mathrm{H}_t|\mathrm{H}_{t-1}) = p_\theta(\boldsymbol{P} \star \mathrm{H}_t|\boldsymbol{P} \star \mathrm{H}_{t-1}) \forall t, \boldsymbol{P}$. Also, because we have chosen the distribution $p_\theta(\mathrm{H}_T|\mathrm{G})$, such that all conditioned part from $\mathcal{B}_{1:i}$ are the same with input G, and all other left elements are sampled from isotrophic noise, the distribution $p_\theta(\mathrm{H}_T|\mathrm{G})$ is also equivariant with $p_\theta(\mathrm{H}_T|\mathrm{G}) = p_\theta(\boldsymbol{P} \star \mathrm{H}_T|\boldsymbol{P} \star \mathrm{G})$. Take these properties back we have

$$\begin{aligned}
p_\theta(\boldsymbol{P} \star \mathrm{H}|\boldsymbol{P} \star \mathrm{G}) &= \int p_\theta(\mathrm{H}_T|\boldsymbol{P} \star \mathrm{G}) p_\theta(\mathrm{H}_1|\boldsymbol{P} \star \mathrm{H}_0) \prod_{t=2}^{T} p_\theta(\mathrm{H}_t|\mathrm{H}_{t-1}) d\mathrm{H}_{1:T} \\
&= \int p_\theta(\boldsymbol{P} \star \mathrm{H}_T|\boldsymbol{P} \star \mathrm{G}) p_\theta(\boldsymbol{P} \star \mathrm{H}_1|\boldsymbol{P} \star \mathrm{H}_0) \prod_{t=2}^{T} p_\theta(\boldsymbol{P} \star \mathrm{H}_t|\boldsymbol{P} \star \mathrm{H}_{t-1}) d\mathrm{H}_{1:T} \\
&= \int p_\theta(\mathrm{H}_T|\mathrm{G}) \prod_{t=1}^{T} p_\theta(\mathrm{H}_t|\mathrm{H}_{t-1}) d\mathrm{H}_{1:T} \\
&= p_\theta(\mathrm{H}|\mathrm{G})
\end{aligned} \tag{33}$$

□

With the conclusion from Eq. (31), we can apply it to Eq. (30)

$$\begin{aligned}
&p_\theta\Big(\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]) \Big| \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) \\
&= \int p_\theta\Big(\boldsymbol{P} \star \big(\mathrm{H}[\mathcal{B}_{1:i-1}] \cup \big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]\big)\big) \Big| \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) d\mathrm{H}[\mathcal{B}_{1:i-1}] \\
&= \int p_\theta\Big(\mathrm{H}[\mathcal{B}_{1:i-1}] \cup \big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]\big) \Big| \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big) d\mathrm{H}[\mathcal{B}_{1:i-1}] \\
&= p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \Big| \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big)
\end{aligned} \tag{34}$$

https://doi.org/10.52202/079017-0229

Apply Eq. (28) and Eq. (34) to Eq. (27), we can finalize our proof that

$$
\begin{aligned}
p_\theta(\boldsymbol{P} \star \mathrm{G}) &= \prod_{i=1}^{K_B} p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \boldsymbol{P} \star \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) p_\theta\Big(\boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]) \;\Big|\; \boldsymbol{P} \star (\mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}])\Big) \\
&= \prod_{i=1}^{K_B} p_\theta\Big(|\mathcal{B}_i| \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big) \\
&= \prod_{i=1}^{K_B} p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\Big) = p_\theta(\mathrm{G})
\end{aligned}
$$

Hence, our method PARD is a permutation-invariant probability model.

A.7 Definition of Graph Transformation

Definition A.1 (Graph Transformation)**.** Given a labeled graph $\mathrm{G} = (\mathrm{V}_G, \mathrm{E}_G)$ and a labeled target graph $\mathrm{H} = (\mathrm{V}_H, \mathrm{E}_H)$ that have the same number of nodes, the graph transformation function $f : \mathrm{G} \to \mathrm{H}$ is defined such that f(G) = H.

Notice that while the graph transformation function requires input and output graph have the same graph size, this function is general enough to accommodate any changing of size operation. Specifically, if G has more nodes than H, the function must assign an 'empty' token to nodes and edges within G that do not correspond to those in H. Conversely, if G has fewer nodes than H, the function will augment G with 'empty' nodes and edges until it matches the size of H. Subsequently, it transforms this augmented version of G to match H.

Definition A.2 (Equivariant Graph Transformation)**.** An equivariant graph transformation f is a graph transformation that satisfies $f(\boldsymbol{P} \star \mathrm{G}) = f(\boldsymbol{P} \star \mathrm{H})$ for any permutation operation $\boldsymbol{P}$.

A.8 Algorithms of Training and Generation

We present blocksize prediction algorithm in Algo. 2 and denoising diffusion algorithm in Algo. 3. Notice while the blocksize network g_θ in Algo. 2 and denoising diffusion network f_θ in Algo. 3 use the same subscript parameter θ, the θ essentially represents concatenation of g and f's parameters.

Algorithm 2 Train blocksize distribution $p_\theta\big(|\mathcal{B}_i| \;\big|\; \mathrm{G}[\mathcal{B}_{1:i-1}]\big)$

Input: G, maximum hop K_h, a network g_θ that takes a graph as input and output graph wise prediction.
Get structural partial order function ϕ of G from Algo.1.
Using ϕ to get the sequence of node blocks $[\mathcal{B}_1, ..., \mathcal{B}_{K_B}]$ for G.
Minimize $\sum_{i=1}^{K_B} \text{CrossEntropy}(g_\theta(\mathrm{G}[\mathcal{B}_i]), |\mathcal{B}_{i+1}|)$, with $|\mathcal{B}_{K_B+1}| = 0$

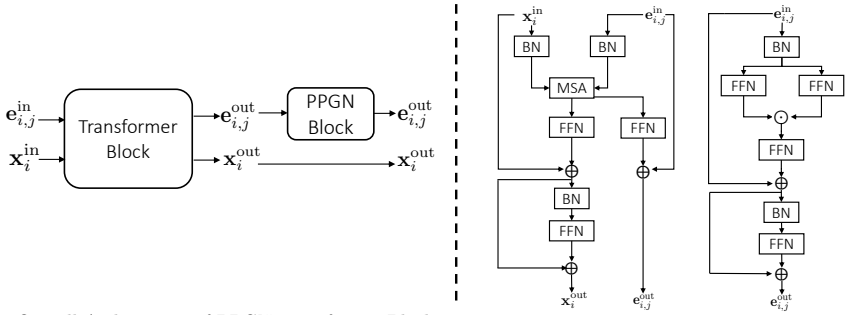
Algorithm 3 Train denoising diffusion for distribution $p_\theta\Big(\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}] \;\Big|\; \mathrm{G}[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big)$

Input: G, max time T, maximum hop K_h, a network f_θ that inputs a graph and outputs nodes and edges predictions.
Get structural partial order function ϕ of G from Algo.1.
Using ϕ to get the sequence of node blocks $[\mathcal{B}_1, ..., \mathcal{B}_{K_B}]$ for G.
Sample $t \sim U(1, ..., T)$
for $i = 1, ..., K_B$ **do**
    $M \leftarrow$ indice mask of $\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]$
    Sample a noise graph $\tilde{\mathrm{G}}[\mathcal{B}_{1:i}]$ from $q_{t|0}(\mathrm{G}[\mathcal{B}_{1:i}])$ according to Eq. (15)
    $\tilde{\mathrm{G}}[\mathcal{B}_{1:i}] \leftarrow M \odot \tilde{\mathrm{G}}[\mathcal{B}_{1:i}] + (1 - M) \odot \mathrm{G}[\mathcal{B}_{1:i}]$
    $X \leftarrow f_\theta(\tilde{\mathrm{G}}[\mathcal{B}_{1:i}]) \odot M$
    $Y \leftarrow \mathrm{G}[\mathcal{B}_{1:i}] \odot M$
    $l_i \leftarrow \mathcal{L}_t(X, Y) + 0.1 * \mathcal{L}_t^{CE}(X, Y)$, using $\mathcal{L}_t$ in Eq. (2) and $\mathcal{L}_t^{CE}$ in Eq. (3).
end for
Minimize $\sum_{i=1}^{K_B} l_i$. (The for loop can be parallelized.)

Algorithm 4 Generation

Input: blocksize model g_θ, diffusion model f_θ; first blocksize distribution from TrainSet.
$\mathrm{G} \leftarrow \emptyset$; $i \leftarrow 1$
Sample n from the first block's size distribution.
while $n > 0$ **do**
  Add a new block $\mathcal{B}_i$ with n nodes into G
  $M \leftarrow$ indice mask of $\mathrm{G}[\mathcal{B}_{1:i}] \setminus \mathrm{G}[\mathcal{B}_{1:i-1}]$
  $\tilde{\mathrm{G}} \leftarrow$ For nodes and edges within M, sample from noise $\boldsymbol{m}_n$ and $\boldsymbol{m}_e$.
  for $j = 1 : T$ **do**
    $\mathrm{p} \leftarrow f_\theta(\tilde{\mathrm{G}})$
    $S \leftarrow$ Sample according to p
    $\tilde{\mathrm{G}} \leftarrow M \odot S + (1 - M) \odot \tilde{\mathrm{G}}$
  end for
  $\mathrm{G} \leftarrow \tilde{\mathrm{G}}$
  $n \leftarrow$ Sample from $g_\theta(\mathrm{G})$
  $i \leftarrow i + 1$
end while
Return: G

A.9 Visualization of the PPGN-Transformer Block



(a) The Overall Architecture of PPGN Transformer Block (b) Transformer Block (c) PPGN Block

Figure 3: The Architecture of the PPGN-Transformer Block. In (b) and (c) we provide illustrations of how edge and node features are processed through Transformer and PPGN blocks.

A.10 Details and Derivations of Causal Matrix-Matrix Product

A.10.1 Derivation of causal version matrix product

For any matrix $\boldsymbol{X}$, let $\boldsymbol{X}_i$ denotes the i-th row of $\boldsymbol{X}$, and $\boldsymbol{X}_{:,i}$ denotes i-th column of $\boldsymbol{X}$. Let $\langle \cdot, \cdot \rangle$ denotes vector inner product. For normal matrix product $\boldsymbol{AB}$, we know that

$$(\boldsymbol{AB})_{ij} = \langle \boldsymbol{A}_i, \boldsymbol{B}_{:,j} \rangle \tag{35}$$

In our causal setting with block ID introduced in main context, the (i, j) position will have its block ID being $\max(\text{block\_ID}(i), \text{block\_ID}(j))$. Hence, the product in Eq. (35) should not use any information outside of block whose ID is larger than $\max(\text{block\_ID}(i), \text{block\_ID}(j))$. Let $\boldsymbol{O}_{ij}$ be the needed safe output at position (i, j) of the matrix-matrix product, one can verify that the following uses all information within useable blocks without any information leakage.

$$\begin{aligned}
\boldsymbol{O}_{ij} &= \langle \boldsymbol{A}_i \odot (\boldsymbol{M}_i \text{ or } \boldsymbol{M}_j), \boldsymbol{B}_{:,j} \rangle \quad \text{(where or denotes elementwise or operation)} \\
&= \langle \boldsymbol{A}_i \odot (\boldsymbol{M}_i + \boldsymbol{M}_j - \boldsymbol{M}_i \odot \boldsymbol{M}_j), \boldsymbol{B}_{:,j} \rangle \\
&= \langle \boldsymbol{A}_i \odot \boldsymbol{M}_i, \boldsymbol{B}_j \rangle + \langle \boldsymbol{A}_i, \boldsymbol{B}_{:,j} \odot \boldsymbol{M}_j \rangle - \langle \boldsymbol{A}_i \odot \boldsymbol{A}_i, \boldsymbol{B}_{:,j} \odot \boldsymbol{M}_j \rangle \\
&= \langle \boldsymbol{A}_i \odot \boldsymbol{M}_i, \boldsymbol{B}_j \rangle + \langle \boldsymbol{A}_i, \boldsymbol{B}_{:,j} \odot \boldsymbol{M}_{:,j}^\top \rangle - \langle \boldsymbol{A}_i \odot \boldsymbol{A}_i, \boldsymbol{B}_{:,j} \odot \boldsymbol{M}_{:,j}^\top \rangle
\end{aligned} \tag{36}$$

Where mask matrix $\boldsymbol{M}$ satisfies $\boldsymbol{M}_{i,j} = 1$ if block_ID(node i) $\geq$ block_ID(node j) else 0.

Based on Eq. (36), we can rewrite it to matrix operation such that

$$\boldsymbol{O} = (\boldsymbol{A} \odot \boldsymbol{M})\boldsymbol{B} + \boldsymbol{A}(\boldsymbol{B} \odot \boldsymbol{M}^\top) - (\boldsymbol{A} \odot \boldsymbol{M})(\boldsymbol{B} \odot \boldsymbol{M}^\top) \tag{37}$$

https://doi.org/10.52202/079017-0229

Where the matrix $\boldsymbol{O}$'s (i, j) position value is just $\boldsymbol{O}_{ij}$ in Eq. (36). Hence we have derived the equation in Eq. (10).

A.10.2 Discussion of expressivity

While the causal matrix-matrix product makes the training more efficient with parallel support, we have to acknowledge that it is highly possible that its expressiveness in graph distinguishing ability is reduced. Hence the causal PPGN should have less expressiveness than the normal PPGN. In fact, we have found that for symmetry-rich datasets like GRID, sequential training of PARD's loss is easier to minimize than parallel training of PARD.

A.10.3 Additional details of parallel training

The previous discussion mainly focuses on preventing information leakage, which is the most critical aspect of causal parallel training. Another essential component of parallel training is the ability to output all predictions for subsequent blocks simultaneously. For GPT, no modifications are necessary for predicting all next tokens, as the next token can simply use the position from the previous token. However, when predicting the next block given all previous blocks, it's not feasible to use any positions within previous blocks to hold the prediction for the next block due to differences in size and lack of alignment. Therefore, as shown in Eq. (6), a virtual block needs to be augmented to serve as a prediction placeholder for the next block. For parallel prediction, a virtual block will be augmented for each block in the graph, with each virtual block having the same block ID as the corresponding original block. To summarize, for a graph with N nodes, it needs to be expanded to $2N$ nodes by adding N virtual nodes for parallel training.

A.11 Experiment Details

Table 7: Dataset summary

Name	**#Graphs**	$\lvert V\rvert_{\text{avg}}$	$\lvert E\rvert_{\text{avg}}$
QM9	133,885	9	19
ZINC250K	249,455	23	50
MOSES	1,936,963	22	47
COMMUNITY-S	200	15.8	75.5
CAVEMAN	200	13.9	68.8
CORA	18,850	51.9	121.2
BREAST	100	55.7	117.0
GRID	100	210	783.0

We use a single RTX-A6000 GPU for all experiments. For discrete diffusion, we follow [52]'s basic discrete diffusion implementation, where exponential moving average is not enabled. For model implementation, we use Pytorch Geometric [14], and we implement our combination of PPGN and Transformer by referencing the code in Maron et al. [30] and Ma et al. [29]. Additionally, we use Pytorch Lightning [12] for training and keeping the code clean. We use Adam optimizer with cosine decay learning rate scheduler to train. For diffusion and blocksize prediction, we also input the embedding of block id and node degree as additional feature, as we have block id computed in preprocessing. Additionally, it is important to observe that nodes within the same block all have the same degree. At diffusion stage, we conditional on the predicted degree for the next block to train diffusion model, where the degree is predicted along with the block size using the same network. We provide source code and all configuration files at https://github.com/LingxiaoShawn/Pard.

A.12 Ablations and Runtime Measure

How does the sampling steps for diffusion models in PARD impact the performance of graph generation? To answer it, we conducted a detailed ablation on number of diffusion steps in QM9, with results in Table. 8. To summarize, too small a number of steps hurt the performance, while too large doesn't help improve the performance further. See the table below. Within the table, we have emphasized a configuration that employs just 140 total diffusion steps. This amount is considerably less than the steps used by DiGress, yet it dramatically outperforms DiGress.

While graphs are trained with a single, fixed generation path based on ϕ, is the same path being used in generation? We acknowledge that although each graph undergoes a single decomposition

Table 8: Comparison of different models on QM9 dataset

Ablation on QM9	PARD					DiGress
Average number of blocks	7	7	7	7	7	1
Number of Diffusion Steps Per Block	10	**20**	40	70	100	500
Total Number of Diffusion Steps	70	**140**	280	490	700	500
Validity	0.9419	0.9708	0.971	0.973	0.9755	95.4
Uniqueness	0.9676	0.9605	0.9579	0.9544	0.9609	97.6
Mol stability	0.8094	0.8627	0.8625	0.8589	0.8561	79.8
Atom Stability	0.9767	0.9847	0.9844	0.9839	0.9837	98.1

block sequence during training, it is possible for the generation path to differ from the training path. This discrepancy is attributed to a phenomenon known as *exposure bias*, a challenge encountered in autoregressive methods and imitation learning alike. Despite this, we have conducted empirical analysis to estimate the likelihood of divergence between the generation and training paths. By sampling 2,000 graphs generated by the model trained on the QM9 dataset, we tracked the generation path for each graph and compared it to the training path defined by the decomposition block sequence algorithm. Our findings indicate that **94.7%** of the generated samples followed the exact same path as their corresponding training path.

Runtime comparison. To show that our method doesn't introduce much runtime overhead, we report the training time in Table 9 used to achieve the number reported on two datasets: QW9 and Moses. The training time for DiGress is mentioned in their github. Notice that the training time is also affected by the machine, we use GPU A6000 in all experiments.

Table 9: Training time comparison on QM9 and Moses datasets

Training Time	QM9	Moses
DiGress	∼6h	∼7 days
PARD	∼12h	∼4 days

A.13 Visualization of Generation

A.13.1 QM9

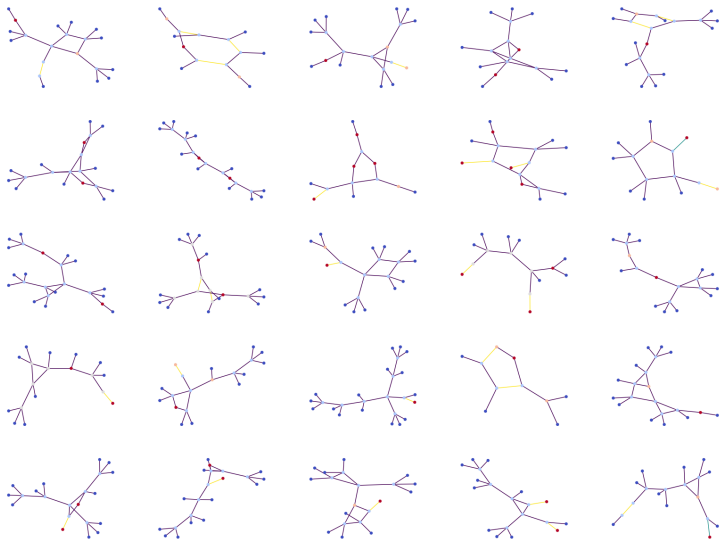


Figure 4: Non curated QM9 (with explicit hydrogens) graphs generated from the PARD trained with 20 steps per block.

https://doi.org/10.52202/079017-0229

GRID	Deg.	Clus.	Orbit
EDP-GNN	0.455	0.238	0.328
GDSS	0.111	0.005	0.070
PARD	0.028	0.002	0.029
PARD with EigenVec.	0.053	0	0.008

Table 10: Performance comparison of diffusion methods on GRID.

A.13.2 Grid

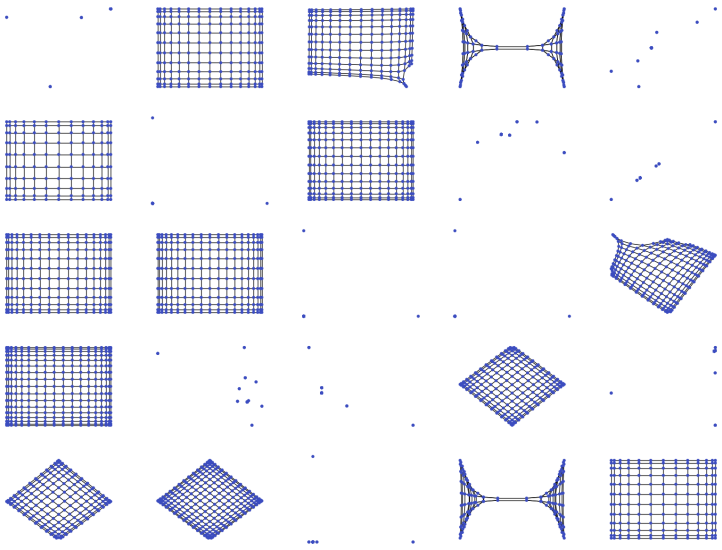


Figure 5: Non curated grid graphs generated from the PARD trained with 50 steps per block.

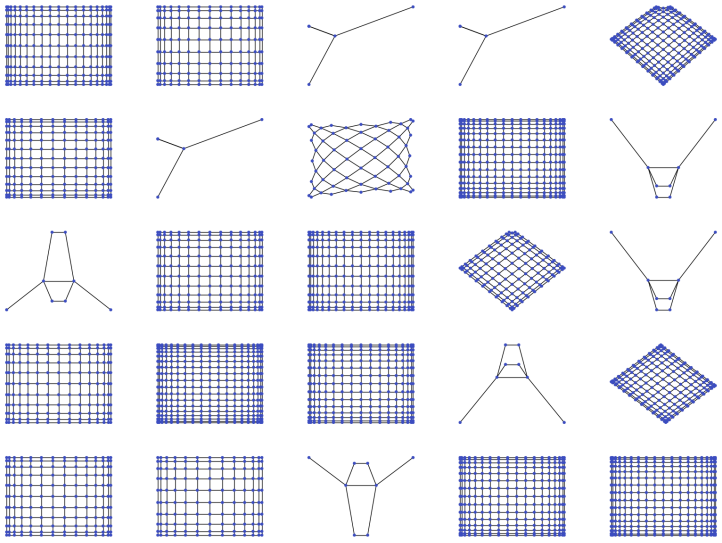


Figure 6: Non curated grid graphs generated from the PARD (with eigenvector) trained with 50 steps per block.

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: We propose a new graph diffusion method that integrates autoregressive graph generation with diffusion. We showcase our method in the main paper with thorough experiments.

   Guidelines:

   - The answer NA 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 NA 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: We briefly discuss the limitations of PARD with respect to long training and sampling times, but we have pointed out potential improvements with for parallel training in causal transformers.

   Guidelines:

   - The answer NA 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**

https://doi.org/10.52202/079017-0229

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

Answer: [Yes]

Justification: All our theorems, formulas and proofs in the paper are numbered and referenced. The assumptions are clearly stated. The proofs are shown in main paper and appendix.

Guidelines:

- The answer NA 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: We introduce a new algorithm and architecture for graph generation tasks. The detailed explanations on the algorithm, and the experiment details are clearly shown in the appendix.

Guidelines:

- The answer NA 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: We have publicized our training repository with an anonymous link referenced in the paper.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We use the same test bed as our benchmarking methods, so the dataset split and evaluation codes are publicly available. The hyperparameter configurations are discussed in the appendix as well as in our code library.

   Guidelines:

   - The answer NA 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: We provide experiment results and evaluations over 10,000-25,000 sampled graphs, so the numbers are of high confidence.

https://doi.org/10.52202/079017-0229

Guidelines:

- The answer NA 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: We provide the memory, running time, sampling time discussions in the main paper and in appendix. All experiments are running on A6000 GPUs.

   Guidelines:

   - The answer NA 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 authors have reviewed the code of ethics.The paper and the code base preserve anonymity.

   Guidelines:

   - The answer NA 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: [NA]

Justification: There is no societal impact of the work performed.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

Answer:[NA]

Justification: The paper poses no such risks.

Guidelines:

- The answer NA 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: All our datasets and benchmarking methods are from publicly available libraries under proper licenses.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

https://doi.org/10.52202/079017-0229

- 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: We provide anonymous code base.

    Guidelines:

    - The answer NA 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: [NA]

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

    Guidelines:

    - The answer NA 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: [NA]

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

    Guidelines:

- The answer NA 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.

https://doi.org/10.52202/079017-0229