Graph Diffusion that can Insert and Delete

Matteo Ninniri
Department of Computer Science
University of Pisa
56127 Pisa (Italy)
matteo.ninniri@phd.unipi.it

Marco Podda
Department of Computer Science
University of Pisa
56127 Pisa (Italy)
marco.podda@unipi.it

Davide Bacciu
Department of Computer Science
University of Pisa
56127 Pisa (Italy)
davide.bacciu@unipi.it

Abstract

Generative models of graphs based on discrete Denoising Diffusion Probabilistic Models (DDPMs) offer a principled approach to molecular generation by systematically removing structural noise through iterative atom and bond adjustments. However, existing formulations are fundamentally limited by their inability to adapt the graph size (that is, the number of atoms) during the diffusion process, severely restricting their effectiveness in conditional generation scenarios such as property-driven molecular design, where the targeted property often correlates with the molecular size. In this paper, we reformulate the noising and denoising processes to support monotonic insertion and deletion of nodes. The resulting model, which we call GRIDDD, dynamically grows or shrinks the chemical graph during generation. GRIDDD matches or exceeds the performance of existing graph Diffusion Models on molecular property targeting despite being trained on a more difficult problem. Furthermore, when applied to molecular optimization, GRIDDD exhibits competitive performance compared to specialized optimization models. This work paves the way for size-adaptive molecular generation with graph diffusion.

1 Introduction

Generating molecules conditioned on predefined structures or properties is a central endeavor in computational chemistry, with applications ranging from *de novo* drug design [You et al., 2018a] to materials discovery [Zhao et al., 2023]. Depending on the conditioning information, we distinguish two key learning tasks: *property targeting*, aimed at generating molecules endowed with prespecified properties; and *property optimization*, where the goal is to edit a given molecule to improve a target property while retaining its core structure. Unlike continuous data, generating graphs must account for their discrete and combinatorial connectivity. For molecules in particular, chemical constraints invalidate most atom-bond and atom-atom combinations, making this problem difficult and, therefore, actively researched. Despite these challenges, deep graph generators [Faez et al., 2021] have achieved striking success in the above-mentioned tasks, due to their ability to approximate complex molecular distributions and their flexibility in incorporating conditioning information.

Based on the way they decode a graph from a latent representation, deep generative models for molecules can be broadly categorized as autoregressive or one-shot [Zhu et al., 2022]. Autoregressive models build the graph sequentially (atom-by-atom as in You et al. [2018b], or fragment-by-fragment

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

 https://doi.org/10.52202/085713-2625

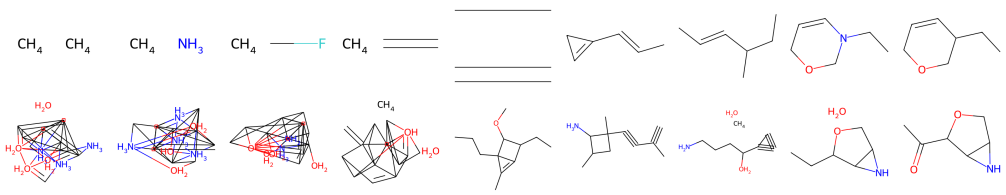


Figure 1: Two qualitative examples of the proposed model (GRIDDD) when generating molecules from the QM9 dataset. In the top row, from left to right, we show a subset of the denoising process to generate a molecule starting from a latent graph with two atoms (which are extremely rare in QM9). GRIDDD successfully inserts six more nodes to obtain a sample resembling the training set's distribution. In the bottom row, we start from a latent with 18 atoms instead (not present in QM9). GRIDDD manages to delete nine atoms and obtain a valid molecule. Notice that, unlike current DDPMs for graphs, the graph size is changed dynamically during denoising.

as in Jin et al. [2018b]), but suffer from order dependence and linearly-scaling sampling time. One-shot methods generate all nodes and edges in a single pass enabling parallel sampling, but struggle to learn high-order interactions. Denoising Diffusion Probabilistic Models (DDPMs) [Ho et al., 2020] for graphs bridge these approaches by progressively removing Gaussian or discrete noise to recover the molecular graph. Indeed, DDPMs preserve permutation invariance and sampling parallelism while iteratively refining long-range interactions (e.g., ring closures), which traditional one-shot decoders are forced to learn in a single transformation. Moreover, DDPMs can be easily adapted to conditional generation with classifier-based [Dhariwal and Nichol, 2021] or classifier-free [Ho and Salimans, 2022] guidance.

One drawback of Diffusion Models (not limited to graphs) is that the sample size remains fixed throughout the generative process. While for different modalities this is less of a concern (e.g., for images it is equivalent to fixing the resolution beforehand), it becomes relevant when generating molecules, and more in general when modeling combinatorial data. In practical terms, this restriction implies that the model *by design* cannot add or remove atoms during generation, leading to two major drawbacks: *a)* in property targeting, it is not possible to make the molecular structure responsive to properties which correlate with the size, e.g., when generating molecules with a specific molecular weight; *b)* in property optimization, the model cannot optimize a given property by adapting the generated structure. Existing methods circumvent this limitation by sampling the graph size before the generative process starts. For example, Vignac et al. [2023a] and Hoogeboom et al. [2022] sample different graph sizes from the empirical distribution of the training dataset, while Ninniri et al. [2024] use an auxiliary classifier to predict the graph size from the conditioning information. Though these workarounds are sufficient for property targeting, they become ineffective for property optimization, where size is strictly related to the structure to optimize. Ketata et al. [2025] adapts to property optimization by sampling additional nodes in the reverse process, but does not incorporate this procedure into the training process.

Motivated by these considerations, our primary contribution is a generalization of the standard discrete diffusion process on graphs. Our approach enables step-wise monotonic node insertions and removals during diffusion, and is specifically designed to change the graph size dynamically throughout the generative process, providing the necessary flexibility to incorporate conditional information more effectively. We implement this novel formulation as a model called GRIDDD (short for Graph Insert-Delete Discrete Diffusion). An intuitive example of how GRIDDD generates molecules is provided in Figure 1. We test GRIDDD on property targeting in two widely used benchmarks (QM9 and ZINC-250k), where it consistently performs on par or better than the state of the art in terms of approximating the target property while keeping high chemical validity, despite having been trained on a more difficult problem. When applied to molecular optimization, GRIDDD convincingly outperforms other molecular optimizers, achieving a higher average improvement and optimization success rate. To the best of our knowledge, this is the first work that allows size-adaptive molecular generation with graph diffusion. Our code is available at https://github.com/mninniri/GrIDDD.

2 Background and related works

Notation We represent molecules with n atoms as graphs $G = (\boldsymbol{X}, \boldsymbol{E})$, where $\boldsymbol{X} \in \mathbb{R}^{n \times a}$ is the matrix of one-hot encoded atom types (over a possible atom types), and $\boldsymbol{E} \in \mathbb{R}^{n \times n \times b}$ is an adjacency (or edge) tensor that encodes both the bond connectivity and the one-hot encoded bond types (over b possible bond types while considering the absence of a bond as an additional bond type). We denote as $\boldsymbol{x}_i \in \mathbb{R}^a$ the i-th row of $\boldsymbol{X}$ and as $\boldsymbol{e}_{ij} \in \mathbb{R}^b$ the slice of $\boldsymbol{E}$ along the first and second dimensions. Oftentimes, we use the terms "nodes" and "edges" to refer to atoms and bonds, respectively. Conditioning vectors will be generally denoted as $\boldsymbol{y} \in \mathbb{R}^d$, d being a positive integer.

2.1 Graph Generative Models for Property Targeting and Optimization

Several models have recently been developed for property targeting, mostly based on graph diffusion since it is particularly suited for conditional generation. For example, DiGress [Vignac et al., 2023a] uses a categorical diffusion process on nodes and edges, and a graph transformer denoiser to learn a generative distribution over molecular graphs. DiGress allows conditional sampling through classifier-based guidance, by training an auxiliary regressor to steer the generation towards the target properties. FreeGress [Ninniri et al., 2024] builds upon DiGress and proposes a classifier-free approach for conditional generation that injects the guide directly into the denoiser during training, greatly improving property-targeting accuracy.

Molecular property optimization is mainly achieved through translation models and optimization models. VJTNNs [Jin et al., 2018b], Seq2Seq models [He et al., 2021], and HierG2G [Jin et al., 2020] all work by translating between different types of molecular representations, ranging from junction trees to SMILES strings. While these models achieve good results, they require a dataset of pairs of similar molecules with specific properties, which are of limited availability. At the same time, they are not designed to generate new molecules from scratch. Optimization models are usually designed for molecular generation. GCPN [You et al., 2018a] and JT-VAE [Jin et al., 2018a] employ different techniques, ranging from reinforcement learning to junction trees, to generate data, and are easily extensible to perform property targeting and optimization.

2.2 Discrete Denoising Diffusion Probabilistic Models

DDPMs consist of an un-parameterized *forward process* $q(\boldsymbol{x}^t|\boldsymbol{x}^{t-1})$ and a parameterized *reverse process* $p_\theta(\boldsymbol{x}^{t-1}|\boldsymbol{x}^t)$. The forward process progressively corrupts the initial data point $\boldsymbol{x}^0$, transforming it into a noise-like sample $\boldsymbol{x}^T$. The reverse process, trained to denoise these samples, aims to reconstruct $\boldsymbol{x}^0$ by sequentially removing the added noise. Sampling from a trained DDPM involves drawing $\tilde{\boldsymbol{x}}^T$ from the noise distribution and applying the reverse process iteratively over T steps to obtain a new sample $\tilde{\boldsymbol{x}}^0 \sim q(\boldsymbol{x})$. While in the original formulation of DDPMs the noise distribution is Gaussian, we focus on *discrete* DDPMs where noise is injected through specially crafted *transition matrices* $\boldsymbol{Q}^t$, with $[\boldsymbol{Q}^t]_{ij} = p(x^t = j|x^{t-1} = i)$, encapsulating the probability of switching from category i to category j at step t of the forward process. Crucially, cumulative transitions from an arbitrary timestep s to t can be expressed as a matrix $\overline{\boldsymbol{Q}}_{t|s} = \prod_{i=s+1}^{t} \boldsymbol{Q}^i$. The reverse process $p_\theta(\boldsymbol{x}^{t-1}|\boldsymbol{x}^t)$ is computed by marginalization over the possible categories of the input $\boldsymbol{x}$.

2.3 Insert and delete operations for Diffusion Models

The idea of insert and delete operations for discrete Diffusion Models originates from Johnson et al. [2021], which, similarly to us, proposed to gradually insert or delete nodes both during the forward and reverse process. However, there are some major differences with our work. Firstly, our solution is designed for graphs, instead of textual data. Secondly, they resort to complex edit summaries to compute the various probability distributions involved, while we simplify the computation of the posteriors by generalizing the denoising process to account for nodes that have yet to be deleted or inserted through the computation. Some continuous-time Diffusion Models [Campbell et al., 2023] can increase the entries in the data point during the reverse process, but can not remove them, making them unsuitable for property optimization. At the same time, they are better suited for continuous data with positional dependencies, rather than graphs.

 https://doi.org/10.52202/085713-2625

3 Graph Insert-Delete Discrete Diffusion

We now introduce our main contribution. The backbone of GRIDDD is a reformulation of the standard discrete denoising diffusion for graphs to support the use of monotonic node insertions and deletions, which we describe in this section.

Preliminaries. Our end goal is to obtain a denoised graph $\boldsymbol{G}^0 = (\boldsymbol{X}^0, \boldsymbol{E}^0)$ with n^0 nodes from a latent noisy graph with $\boldsymbol{G}^T = (\boldsymbol{X}^T, \boldsymbol{E}^T)$ with n^T nodes. To change size from n^T to n^0, we *monotonically* delete (if $n^T > n^0$) or insert (if $n^T < n^0$) $|\Delta^T|$ nodes, where $\Delta^T = n^T - n^0$, as the reverse process advances. This gives us full control over the graph size at any given moment. The timesteps when insertions and deletions are performed are sampled from the normalization of the absolute value of the logistic distribution's probability density function $\zeta^{'}(t)$:

$$\zeta^{'}(t) = |\frac{e^{\frac{t-D}{w}}}{w(e^{\frac{t-D}{w}}+1)^2}|. \tag{1}$$

The function is scaled in a way such that $\sum_{t=0}^{T} \zeta^{'}(t) = 1$, with $\zeta^{'}(0) = \zeta^{'}(T) = 0$. The parameter D indicates the timestep in which the function is maximized, while w is the scale parameter (from a visual standpoint, it controls the steepness of the curve). During training, the final size n^T that a training sample with size n^0 will assume is sampled from a discrete distribution on the number of nodes $h_{n^0}(n)$. In our work, we defined it as follows, and then normalized it to sum to one:

$$h_{n^0}(n) = p_{\text{max}} + \frac{p_{\text{min}} - p_{\text{max}}}{n_{\text{max}}}|n - n^0|, \tag{2}$$

where n_{max} is the maximum size that a graph may assume (which can be set to be higher than the size of the largest graph in the dataset), while p_{min} and p_{max} are hyperparameters. Intuitively, the highest probability p_{max} occurs when $n = n^0$, and it linearly decreases to p_{min} with a rate equal to $\frac{p_{\text{min}} - p_{\text{max}}}{n_{\text{max}}}$ as the absolute difference between n^0 and n increases.

3.1 Forward process

The discussion above implies that we have three different forward processes, depending on whether $\Delta^T > 0$, $\Delta^T < 0$, or $\Delta^T = 0$. We start by describing the latter, which is the simplest and serves as a basis for the deletion and insertion cases. Building on previous works [Vignac et al., 2023a, Ninniri et al., 2024], when $\Delta^T = 0$, the forward process is implemented as:

$$q(\boldsymbol{G}^t \mid \boldsymbol{G}^{t-1}) = (\boldsymbol{X}^{t-1}\boldsymbol{Q}^t_{\boldsymbol{X}}, \boldsymbol{E}^{t-1}\boldsymbol{Q}^t_{\boldsymbol{E}}). \tag{3}$$

Focusing on the nodes $\boldsymbol{X}$ (edges are defined analogously), the associated transition matrix is:

$$\boldsymbol{Q}^t_{\boldsymbol{X}} = \alpha^t \boldsymbol{A}_{\boldsymbol{X}} + (1 - \alpha^t)\boldsymbol{B}_{\boldsymbol{X}}, \tag{4}$$

where $\boldsymbol{A}_{\boldsymbol{X}} = \boldsymbol{I}$, $\boldsymbol{B}_{\boldsymbol{X}} = \mathbf{1}_a \boldsymbol{m}_{\boldsymbol{X}}^{\intercal}$, with $\boldsymbol{m}_{\boldsymbol{X}}$ being the vector of the marginal distribution of the atom types computed on the training set, and $\{\alpha_t\}_{i=1\ldots T}$ being a set of *noise schedulers*, chosen such that $\alpha_1 = 1$, $\alpha_T = 0$, and the intermediate elements gradually decrease from one to zero. Multi-step transitions from an arbitrary timestep s can be computed as a matrix $\overline{\boldsymbol{Q}}^{t|s}_{\boldsymbol{X}}$ defined as:

$$\overline{\boldsymbol{Q}}^{t|s}_{\boldsymbol{X}} = \prod_{i=s+1}^{t} \boldsymbol{Q}^i_{\boldsymbol{X}} = \overline{\alpha}^{t|s}\boldsymbol{A}_{\boldsymbol{X}} + (1 - \overline{\alpha}^{t|s})\boldsymbol{B}_{\boldsymbol{X}}, \tag{5}$$

where $\overline{\alpha}^{t|s} = \frac{\overline{\alpha}^{t|0}}{\overline{\alpha}^{s|0}}$, with $\overline{\alpha}^{t|0} = \prod_{i=1}^{t} \alpha^i = \cos\left(\frac{\pi}{2}\frac{(\frac{t}{T}+s)^{\nu}}{1+s}\right)^2$ (notice the hyperparameter ν).

Monotonic deletions ($\Delta^T < 0$). To introduce deletions, we build on Johnson et al. [2021] and treat them as an additional atom type `DEL`. This is similar to absorbing diffusion [Kong et al., 2023], in which the forward process consists of progressively switching all atom types to the deletion type; our strategy is more general, as we allow for nodes and edges to change category during the forward and reverse processes. During the reverse process, nodes need to be reinserted at step t whenever they were deleted during the forward process at the same timestep (that is, if they transitioned to type `DEL` at forward step t). Consequently, such nodes are expected to revert to a

$$\boldsymbol{A}^* = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & & & \texttt{D} & \texttt{D}^* \end{array} \\ \left[\begin{array}{cccccc} & & & & 0 & 0 \\ & \boldsymbol{A} & & & \vdots & \vdots \\ & & & & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 & \\ 0 & \cdots & 0 & 1 & 0 & \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \\ \texttt{D} \\ \texttt{D}^* \end{array} \end{array} \quad (1) \qquad \boldsymbol{B}^* = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & & & \texttt{D} & \texttt{D}^* \end{array} \\ \left[\begin{array}{cccccc} & & & & 0 & 0 \\ & \boldsymbol{B} & & & \vdots & \vdots \\ & & & & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 & \\ 0 & \cdots & 0 & 1 & 0 & \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \\ \texttt{D} \\ \texttt{D}^* \end{array} \end{array} \quad (2)$$

$$\boldsymbol{C}^* = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & & & \texttt{D} & \texttt{D}^* \end{array} \\ \left[\begin{array}{cccccc} 0 & \cdots & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 1 \\ 0 & \cdots & 0 & 1 & 0 & \\ 0 & \cdots & 0 & 1 & 0 & \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \\ \texttt{D} \\ \texttt{D}^* \end{array} \end{array} \quad (3) \qquad \boldsymbol{D}^* = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & & & \texttt{D} & \texttt{D}^* \end{array} \\ \left[\begin{array}{cccccc} 0 & \cdots & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & 0 & 1 & 0 & \\ 0 & \cdots & 0 & 1 & 0 & \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \\ \texttt{D} \\ \texttt{D}^* \end{array} \end{array} \quad (4)$$

Figure 2: The matrices employed in the computation of $\boldsymbol{Q}^{*t}$ and $\overline{\boldsymbol{Q}}^{*t|s}$. For spacing reasons, the states DEL and DEL* have been shortened as D and D*.

proper (non-DEL) type at the next step $t-1$. However, since the DEL state is absorbing (meaning $p(\boldsymbol{x}^t \neq \texttt{DEL} \mid \boldsymbol{x}^{t-1} = \texttt{DEL}) = 0$), such transition is forbidden: once a node is deleted, it must remain deleted up to the end of the forward process, as doing otherwise would violate monotonicity. To enable node reinsertion, we introduce an auxiliary transient type DEL*, designed such that $p(\boldsymbol{x}^t = \texttt{DEL} | \boldsymbol{x}^{t-1} = \texttt{DEL}^*) = 1$, i.e., the transition from DEL* to DEL during the forward process is deterministic, and $p(\boldsymbol{x}^{t-1} \in \{\texttt{DEL}, \texttt{DEL}^*\} | \boldsymbol{x}^t = \texttt{DEL}^*) = 0$. This ensures that if a node is in state DEL* during the reverse process, it can only switch to a proper (non-DEL) type. This mechanism guarantees that nodes deleted at step t can be reinserted at step $t-1$, thereby preventing the absorbing DEL state from blocking node recovery. To ensure that the final graph size is consistent with the one fixed at the start of the forward process, preventing that too many nodes switch into the deletion state, we employ a hybrid scheme, in which $n_0 - |\Delta^T|$ nodes undergo the standard forward process (described by Eq. 4), while $|\Delta^T|$ nodes use a different transition matrix[1] defined as:

$$\boldsymbol{Q}^{*t} = \zeta(t) \left(\alpha^t \boldsymbol{A}^* + (1-\alpha^t)\boldsymbol{B}^*\right) + (1-\zeta(t))\,\boldsymbol{C}^*. \tag{6}$$

Above, $\boldsymbol{A}^*$ and $\boldsymbol{B}^*$ are $\boldsymbol{A}$ and $\boldsymbol{B}$ augmented to account for the deletion types, $\boldsymbol{C}^*$ describes the transition from a normal type to a deletion (see Figure 2), and $\zeta(t)$ is the integral of $\zeta^{'}(t)$ described in Equation 1. Since $\zeta^{'}(t)$ is a probability distribution, $\zeta(t)$ can be seen as its cumulative distribution function; therefore, at time T, $\boldsymbol{Q}^{*T} = \boldsymbol{C}^*$. If $\zeta^{'}$ is chosen such that $\zeta(T-1) = 0$, we enforce exactly $|\Delta^T|$ nodes with DEL type at timestep T. We refer the reader to Appendix A.1 for more details on the function $\zeta^{'}$ used in this study. Computing the cumulative transition matrix $\overline{\boldsymbol{Q}}^{*t|s} = \prod_{i=s+1}^{t} \boldsymbol{Q}^{i*}$ requires an additional matrix $\boldsymbol{D}^*$ (also displayed in Figure 2):

$$\overline{\boldsymbol{Q}}^{*t|s} = \overline{\zeta}^{t|s} \left(\overline{\alpha}^{t|s}\boldsymbol{A}^* + \left(1-\overline{\alpha}^{t|s}\right)\boldsymbol{B}^*\right) + \overline{\zeta}^{t-1|s}\,(1-\zeta(t))\,\boldsymbol{C}^* + \left(1-\overline{\zeta}^{t-1|s}\right)\boldsymbol{D}^*, \tag{7}$$

where $\overline{\zeta}^{t|s} = \prod_{i=s+1}^{t} \zeta(i)$. Intuitively, $\boldsymbol{D}^*$ represents a state where all nodes are immediately switched to state DEL. If a node is set to category DEL or DEL*, so are the edges associated with it.

Monotonic insertions ($\Delta^T > 0$). Node insertions are defined implicitly (i.e., without a specific INS type) since by design, a node is effectively inserted in the graph, or "activated", only at timestep $s+1$ if the model sampled s as insertion time. The critical operation after inserting nodes is to establish which category they belong to. Our solution is to sample the inserted node's state from the marginal distribution $\boldsymbol{m}_{\boldsymbol{X}}$ of the graph itself, which can be precomputed with no significant computational overhead (for more information, refer to Appendix A.2). Once a node is inserted, so are the edges connecting it to the rest of the graph. Their initial state is computed analogously using $\boldsymbol{m}_{\boldsymbol{E}}$.

[1]From here on, we drop the subscript $\boldsymbol{X}$ for brevity. Equations 6–9 are defined analogously for $\boldsymbol{E}$.

https://doi.org/10.52202/085713-2625

3.2 Reverse process

The reverse process starts from a randomly initialized graph and gradually removes noise. However, since our main objective is to perform conditional generation, we also input the conditioning vector $\boldsymbol{y}$ using classifier-free guidance [Ho and Salimans, 2022]. We factorize the reverse process as the product of the individual reverse probabilities of the nodes and edges (assuming mutual independence, which we discuss in Appendix B.4):

$$p_\theta(\boldsymbol{G}^{t-1}|\boldsymbol{G}^t, \boldsymbol{y}) = \prod_{i\in 0,\dots,n^t-1} p_\theta(\boldsymbol{x}_i^{t-1}|\boldsymbol{x}_i^t, \boldsymbol{y}) \prod_{i,j\in 0,\dots,n^t-1} p_\theta(\boldsymbol{e}_{ij}^{t-1}|\boldsymbol{e}_{ij}^t, \boldsymbol{y}). \tag{8}$$

In previous works [Ninniri et al., 2024], the posterior for the nodes was implemented with the aid of a neural network $p_\theta(\boldsymbol{x}^0 = \boldsymbol{x}|\boldsymbol{x}^t, \boldsymbol{y})$ tasked to predict the true atom and bond types given their noisy categories and the guide. However, this no longer works in our setting, as we cannot predict nodes that did not exist when $t = 0$ but were inserted subsequently during the forward process. To address this issue, we let the main model predict, alongside $\boldsymbol{G}^0$, the activation time $\hat{s}$ of each node and introduce the following generalized posterior:

$$p_\theta(\boldsymbol{x}^{t-1}|\boldsymbol{x}^t, \boldsymbol{y}) = \sum_{\boldsymbol{x}\in\chi} \frac{q(\boldsymbol{x}^t|\boldsymbol{x}^{t-1})q(\boldsymbol{x}^{t-1}|\boldsymbol{x}^{\hat{s}} = \boldsymbol{x})}{q(\boldsymbol{x}^t|\boldsymbol{x}^{\hat{s}} = \boldsymbol{x})} p_\theta(\boldsymbol{x}^{\hat{s}} = \boldsymbol{x}|\boldsymbol{x}^t, \boldsymbol{y}). \tag{9}$$

The activation time of the nodes belonging to the original graph is assumed to be zero. Notice that when $q(\boldsymbol{x}^t|\boldsymbol{x}^0 = \boldsymbol{x}) = 0$, then $p_\theta(\boldsymbol{x}^{t-1}|\boldsymbol{x}^t, \boldsymbol{y})$ is set to zero as well. The edge posterior is computed analogously. Below, we describe how to extend the base setting to support insertions and deletions.

Deletions. During the reverse process, nodes deleted during the forward process need to be re-inserted. We do so with an auxiliary neural network $g_\phi(\boldsymbol{G}^t)$ that predicts, given the latent noisy graph $\boldsymbol{G}^t$, the number of DEL*s to add before passing the model to the main neural network that predicts the original graph $\boldsymbol{G}^0$ given $\boldsymbol{G}^t$. The auxiliary network is trained alongside the main model by masking out the nodes set to DEL* in $\boldsymbol{G}^t$ after computing $p_\theta(\boldsymbol{G}^0|\boldsymbol{G}^t, \boldsymbol{y})$, which reduces the training time significantly. Notice that g_ϕ is trained only when $\zeta^{'}(t) > 0$, since it is not possible to insert nodes otherwise.

Insertions. Conversely to the deletion case, nodes inserted during the forward process need to be deleted. To do this, we let the main model predict, alongside $\boldsymbol{G}^0$, the activation time $\hat{s}$ of each node. Then, at timestep t, we remove from the graph the nodes predicted to have been inserted at step t. The activation time of the nodes belonging to the original graph is assumed to be zero.

3.3 Training and sampling

Training. During the learning phase, our goal is to train the model by corrupting each dataset sample $\boldsymbol{G}^0 = (\boldsymbol{X}^0, \boldsymbol{E}^0)$ into $\boldsymbol{G}^{*t} = (\boldsymbol{X}^{*t}, \boldsymbol{E}^{*t})$, with $t \leq T$, sampled from a uniform distribution, using the forward process. Having been subject to insertions or deletions, $\boldsymbol{X}^{*t}$ might contain only a subset of the nodes in $\boldsymbol{X}^0$ (if we deleted nodes) or include a new set of nodes (the ones that have been inserted up to step t). We train the main model to predict the values that the nodes $\boldsymbol{X}^{*t}$ and edges $\boldsymbol{E}^{*t}$ had at their respective activation times $\boldsymbol{S}^*$ (which may be equal to zero if the element belongs to the original graph) and denote these prediction as $\widehat{\boldsymbol{X}}^{*0}$ and $\widehat{\boldsymbol{E}}^{*0}$. At the same time, we predict the activation times themselves, which we denote as $\widehat{\boldsymbol{S}}^*$. Finally, we train the auxiliary model $g_\phi(\boldsymbol{G}^{*t})$ to predict the correct amount $n_{\texttt{DEL}^*}$ of DEL*s that were present in $\boldsymbol{G}^{*t}$, which we denote as $\widehat{n}_{\texttt{DEL}^*}$. The final loss is a sum of Cross-Entropy (CE) terms of the targets $\boldsymbol{X}^{*t}, \boldsymbol{E}^{*t}, \boldsymbol{S}^*$, and $n_{\texttt{DEL}^*}$ against their respective predictions:

$$\lambda_X \mathrm{CE}(\widehat{\boldsymbol{X}}^{*0}, \boldsymbol{X}^{*0}) + \lambda_E \mathrm{CE}(\widehat{\boldsymbol{E}}^{*0}, \boldsymbol{E}^{*0}) + \lambda_S \mathrm{CE}(\widehat{\boldsymbol{S}}^*, \boldsymbol{S}^*) + \lambda_{\texttt{DEL}} \mathrm{CE}(\widehat{n}_{\texttt{DEL}^*}, n_{\texttt{DEL}^*}), \tag{10}$$

where $\lambda_X, \lambda_E, \lambda_E$, and $\lambda_{\texttt{DEL}}$ are hyperparameters that control the weight of each term. If we are training a model to condition on a specific set of properties $\boldsymbol{y}$, it is also fed at training time an auxiliary term $\boldsymbol{c}$ which, with a probability equal to $1 - \rho$, ρ being an hyperparameter, will be equal to $\boldsymbol{y}$, while in the remaining cases it will assume the value of a parametrized placeholder $\overline{\boldsymbol{y}}$, which is randomly initialized. This technique is called *conditional dropout* and is shown to produce better results in conditional generation [Ho and Salimans, 2022]. Algorithm 1 in Appendix A.3 describes the new training process.

Sampling. During sampling, we start from a random latent $\boldsymbol{G}^T$ sampled from the marginal distributions on the nodes and edges $\boldsymbol{m_X}$ and $\boldsymbol{m_E}$. The size of a sample is also chosen from the marginal distribution of the nodes in the training set. At each step, we first insert as many $\texttt{DEL}^*$ nodes as predicted by the auxiliary neural network. Then, we feed the current latent $\boldsymbol{G}^t$ to the main model to predict $\boldsymbol{G}^{*0}$. That is, the values that each node and edge had at their respective activation timesteps, as well as the activation timesteps themselves. Afterwards, we delete the nodes that are predicted to have been inserted at timestep t. Finally, we use these information to compute Equation 9, and sample from it the node and edge values at step $t-1$ to obtain $\boldsymbol{G}^{t-1}$. We then repeat the operations using it as the new $\boldsymbol{G}^t$, and we keep doing so for a total of T steps until we obtain the final graph $\boldsymbol{G}^0$. In the case in which we are performing conditional generation on a property $\boldsymbol{y}$, the term $p_\theta(\boldsymbol{x}^{\hat{s}}|\boldsymbol{x}^t)$ is re-elaborated as follows:

$$p_\theta(\boldsymbol{x}^{\hat{s}} = \boldsymbol{x}|\boldsymbol{x}^t, \boldsymbol{y}) = p_\theta(\boldsymbol{x}^{\hat{s}} = \boldsymbol{x}|\boldsymbol{x}^t, \overline{\boldsymbol{y}}) + \lambda(p_\theta(\boldsymbol{x}^{\hat{s}} = \boldsymbol{x}|\boldsymbol{x}^t, \boldsymbol{y}) - p_\theta(\boldsymbol{x}^{\hat{s}} = \boldsymbol{x}|\boldsymbol{x}^t, \overline{\boldsymbol{y}})), \tag{11}$$

where λ is a hyperparameter controlling the influence of the conditioning vector $\boldsymbol{y}$. Algorithm 2 in Appendix A.3 describes the sampling process in detail.

4 Experiments

Here, we detail the experiments to evaluate GRIDDD on different property targeting and optimization tasks. Our code is based on MiDi [Vignac et al., 2023b] which, in turn, is based on DiGress. In all our experiments, we have $T = 500$ diffusion timesteps. The function $\zeta^{'}(t)$ is parameterized with $w = 0.05$ and $D = \frac{T}{2} = 250$. Similarly to MiDi, we use $\nu = 1$ for the node matrices' noise schedulers and $\nu = 1.5$ for the edge matrices. λ_X and λ_E are set, respectively, to 1 and 2. We set the hyperparameters p_{min} and p_{max} in Equation 2 respectively as 0.2 and 1.

Datasets. Following previous works [Ninniri et al., 2024, Vignac et al., 2023a], we used QM9 [Ramakrishnan et al., 2014], a dataset of 133k molecules made by up to 9 non-hydrogen atoms, and ZINC-250k, a collection of 250k drug-like molecules selected from the ZINC dataset [Irwin and Shoichet, 2005]. On ZINC-250k, the molecules were first preprocessed by removing stereochemistry information and infrequent non-neutrally-charged atoms, leaving only N+ and O-. These atoms have been treated as standalone atom types. Dataset splits are described in Appendix B.1.

4.1 Property Targeting

We adhere to the setup of Ninniri et al. [2024], where they extract 100 property vectors from the test set and generate, for each one of them, 10 molecules. Then, they compute the Mean Absolute Error (MAE) between the target properties $\boldsymbol{y}$ and the estimated properties of the generated molecules $\widehat{\boldsymbol{y}}$:

$$\text{MAE}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \frac{1}{1000} \sum_{i=1}^{100} \sum_{j=1}^{10} |\boldsymbol{y}_i - \widehat{\boldsymbol{y}}_{i,j}|. \tag{12}$$

On QM9, we targeted the Dipole Moment μ and the Highest Occupied Molecular Orbital (HOMO) properties, while on ZINC-250k we targeted the Log-Partition coefficient (LogP), the Quantitative Estimation of Drug-likeness (QED), and the molecular weight (MW). Since MW strongly correlates with the number of nodes of the molecular graph, it provides a natural and intuitive testbed to assess the ability to flexibly adapt the graph size, which is precisely what GRIDDD was designed for. Our baseline for comparison is discussed in Appendix B.3.

4.2 Property Optimization

We use the test suite by Jin et al. [2020], which consists of three experiments on the ZINC-250k dataset. The objective is to generate molecules that improve on a certain property while meeting a similarity constraint. To optimize a molecule, we corrupt it for 100 steps before starting the denoising process, setting the property vector $\boldsymbol{y}$ as the target value that the optimized molecule should have. In the **LogP** experiment, we optimize the 800 molecules with the lowest LogP among those in the test set. For each of them, we sample 20 candidates starting from different latents and only keep the one with the largest improvement among the ones with a fingerprint Tanimoto similarity with the starting molecule equal to or above a threshold δ. We report the average of these improvements

 https://doi.org/10.52202/085713-2625

across the 800 molecules, with $\delta \in \{0.4, 0.6\}$. In the **QED** experiment, we optimize 800 molecules from the test set with QED in the range $[0.7, 0.8]$. For each molecule, we sample 20 candidates from different latents. We consider the optimization successful if at least one among the candidates has a QED in the range $[0.9, 1.0]$ and a fingerprint Tanimoto similarity with the starting molecule ≥ 0.4. We report the success rate across the 800 molecules. In the **DRD2** experiment, we optimize for the biological activity against the dopamine type 2 receptor, selecting 800 molecules from the test set with DRD2 activity score ≤ 0.05. For each molecule, we sample 20 candidates from different latents. We consider the optimization successful if at least one among the candidates has a DRD2 score ≥ 0.5 and a fingerprint Tanimoto similarity with the starting molecule ≥ 0.4. We report the success rate across the 800 molecules. Our baseline for comparison is discussed in Appendix B.3.

4.3 Out-of-distribution sampling

To assess whether GRIDDD is able to generate valid molecules outside the training distribution, we train an instance of DiGress and one of GRIDDD to generate samples unconditionally on QM9. However, during training, we force GRIDDD to *insert* up to 14 nodes in the molecular graph, despite the fact that QM9 features molecules with up to 9 atoms. Then, during generation, we force both models to generate molecules with up to 15 atoms (that is, above GRIDDD's training capacity).

Experimental setup. The denoising network shares a similar architecture to FreeGress, and is described in Appendix B.2. For QM9, we use the same hyperparameters employed by the authors. The auxiliary neural network used to predict the number of DEL*s also uses the exact same setup, with the exception that it only uses one layer. For ZINC-250k, we also use the same hyperparameters as FreeGress except for the number of layers, which we set to 10 instead of the original 12 to use approximately the same number of parameters. The size of the linear layer used to predict the activation time is 256. In QM9, we set the guidance scale to $\lambda = 3$, while in ZINC-250k we set $\lambda = 2$. These values have been chosen since they were the ones offering the best results for FreeGress. All experiments have been performed on an nVidia A100 GPU with 80 GBs of VRAM (two on ZINC-250k).

5 Results

Here, we detail the experimental results obtained by GRIDDD. A visual collection of samples obtained from the model is available in Appendix D.3

5.1 Property targeting

QM9. Results are displayed in Table 1. As can be seen, GRIDDD stably achieves state-of-the-art performance. When conditioning on μ, it improves MAE by 14% at the expense of a slightly lower validity. Conversely, when conditioning on HOMO, the model achieves a better validity at the expense of a slightly higher MAE. For ablation purposes, we also include an unconditional model in the comparison, which is unable to optimize the properties to a satisfactory degree. A discussion of the failure cases in QM9 is presented in Appendix D.4.

Table 1: Results of property targeting on QM9.

Method	μ		**HOMO**	
	MAE $\downarrow$	**Val.** $\uparrow$	**MAE** $\downarrow$	**Val.** $\uparrow$
Unconditional	1.68 ± 0.15	91.5 %	0.95 ± 0.10	91.5 %
DiGress	0.80 ± 0.07	82.5 %	0.61 ± 0.07	91.2 %
FreeGress	0.74 ± 0.08	**83.7** %	$\mathbf{0.32 \pm 0.04}$	90.1 %
GRIDDD	$\mathbf{0.66 \pm 0.07}$	79.0 %	0.37 ± 0.07	**94** %

ZINC-250k. Results are shown in Table 2. On ZINC-250k, GRIDDD once again achieves comparable or better MAE, while improving validity rates in two cases out of three. Of particular interest is the MW experiment, as this property strongly correlates with the graph size. Note that FreeGress'

results have been obtained using an auxiliary neural network to predict the optimal number of atoms the molecule should have given the target MW, information which is used to set the graph's size before starting the denoising process. GRIDDD is capable of halving FreeGress' MAE without any prior knowledge of the graph size distribution. In summary, GRIDDD shows an improved capacity in generating valid molecules without sacrificing the MAE. What is truly relevant in these results is the fact that GRIDDD is capable of obtaining them despite being trained on a more difficult task, showing that our solution has a smoother tradeoff between validity and accuracy.

Table 2: Results of property targeting on the ZINC-250k dataset.

Method	LogP		QED		MW	
	MAE ↓	Val. ↑	MAE ↓	Val. ↑	MAE ↓	Val. ↑
Unconditional	1.52 ± 0.12	86.1 %	0.15 ± 0.01	86.1 %	74.16 ± 6.71	86.1 %
DiGress	0.74 ± 0.08	74.6 %	0.15 ± 0.01	85.1 %	20.92 ± 2.90	40.4 %
FreeGress	$\mathbf{0.17 \pm 0.01}$	84.9 %	0.04 ± 0.01	84.9 %	8.96 ± 1.93	79.7 %
GRIDDD	0.19 ± 0.02	**87.9** %	$\mathbf{0.04 \pm 0.00}$	**87.2** %	$\mathbf{4.89 \pm 0.57}$	**84.2**%

5.2 Out-of-distribution sampling

Results are displayed in Figure 3. While both models perform similarly with up to 12 atoms, the validity rates of DiGress steeply decrease on larger graph sizes. Notably, GRIDDD still generates 35% of valid molecules with 15 nodes, despite being trained on latents with up to 14 nodes. This shows how our model is fairly usable in out-of-distribution sampling, as it is able to learn graph sizes that do not appear in the training dataset.

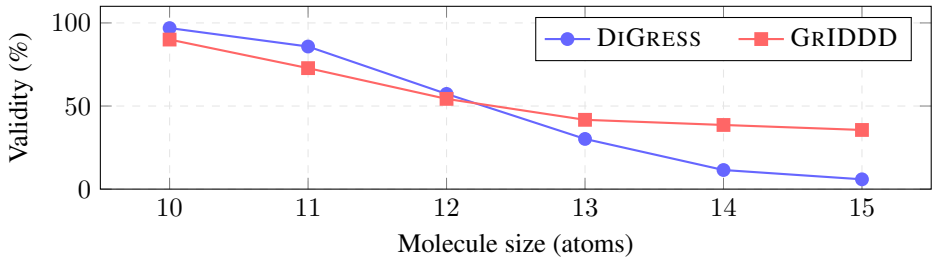


Figure 3: Validity results of out-of-distribution sampling on QM9.

5.3 Property optimization

Results are summarized in Table 3. Regarding the LogP property optimization, GRIDDD outperforms all competitors in both settings in terms of improvement. In particular, when the similarity threshold is 0.6, it achieves an average improvement almost twice as high as the second best. In the other two tasks, GRIDDD scores remarkably better, especially in the QED task where it improves 5 times the success rates of GCPN. An additional comparison with the method by Ketata et al. [2025] (under a slightly different experimental setup) is provided in Appendix D.1, showing that GRIDDD performs comparably or better than methods that use 3D information to optimize molecules.

5.4 Computational considerations

The insertions and deletions performed by GRIDDD inevitably introduce computational overhead during training, leading to slightly increased training durations compared to methods that do not employ these operations. In particular, our analysis indicates that it is 30% slower than FreeGress (see Appendix C.1 for an extended discussion). However, its flexibility in adapting the number of denoising steps to the task makes it generally faster to sample from (as detailed in Appendix C.2).

Ablation. To test the effective influence of insert and delete operations over the optimization process, we performed the same experiments with GRIDDD but disabling insertions and deletions.

https://doi.org/10.52202/085713-2625

Table 3: Benchmark on graph property optimization.

Method	LogP (sim $\geq$ 0.4)		LogP (sim $\geq$ 0.6)		QED (sim $\geq$ 0.4)		DRD2 (sim $\geq$ 0.4)	
	Improv. $\uparrow$	**Div.**$\uparrow$	**Improv.** $\uparrow$	**Div.**$\uparrow$	**Succ.**$\uparrow$	**Div.**$\uparrow$	**Succ.**$\uparrow$	**Div.**$\uparrow$
JT-VAE	1.03 ± 1.39	-	0.28 ± 0.79	-	8.8 %	-	3.4 %	-
CG-VAE	0.61 ± 1.09	-	0.25 ± 0.74	-	4.8 %	-	-	-
GCPN	2.49 ± 1.30	-	0.79 ± 0.63	-	9.4 %	0.216	4.4 %	**0.152**
GRIDDD	$\mathbf{2.70 \pm 0.94}$	0.482	$\mathbf{1.33 \pm 0.61}$	0.280	**45.1** %	**0.283**	**5.0** %	0.121

The results show that while the success rate remains relatively unchanged in the LogP and DRD2 experiments, it significantly drops to 33.8% when optimizing QED, likely because the QED score is a function of the molecular weight (and thus, it correlates with the number of atoms). We conclude that, similarly to targeting MW, controlling the graph size substantially increases success rates.

A naive form of insertion and deletion operations can be achieved implicitly on any graph DDPM by working with a large graph size (even beyond the largest in the training data), treating excess nodes/edges as padding nodes marked by a special `PAD` class which allows their removal at the end of the denoising process. Clearly, this method is slower to train and sample from, as it always works with maximally sized graphs. To check that this approach is also less accurate, we compared two DiGress variants that implement node and edge padding against GRIDDD: the first only assigns the `PAD` category to padding nodes, while the second also assigns it to their associated edges. All models were trained on the QM9 dataset for 250 epochs, after which we sampled 100 molecules each without conditioning. The results are displayed in Table 4. As it is possible to see, while the baseline generates 3% more valid molecules than GRIDDD, they severely underperform in every other metric.

Table 4: Comparison of GrIDDD with DiGress-based padding variants that perform implicit insertions and deletions. Val: validity, Avg NC: average number of connected components, Max NC: maximum number of connected components, NSC: number of graphs sampled with a single connected component, XCE (ECE): validation cross-entropy over nodes (edges).

Model	**Val** $\uparrow$	**Avg NC** $\downarrow$	**Max NC** $\downarrow$	**NSC** $\uparrow$	**XCE** $\downarrow$	**ECE** $\downarrow$
GrIDDD	0.97	**1**	**1**	**100**	**0.44**	**0.32**
Nodes `PAD`	**1**	1.56	8	69	0.47	0.37
All `PAD`	**1**	1.2	4	84	0.48	0.38

In Appendix D.2, we conduct a sensitivity analysis on some critical hyperparameters of GRIDDD, showing that results are robust to their choices.

6 Conclusions

We introduced GRIDDD, a discrete DDPM which supports the insertion and deletion of atoms during the denoising process. This addresses one of the major limitations of graph DDPMs, which cannot change the graph size throughout the generative process. In future studies, we aim to study a simplified approach that does not need an auxiliary network to predict when to insert a node during the denoising process, and whether the distribution of the inserted nodes can be learned rather than employing predefined heuristics. Lastly, since the methodology we propose is quite general, we foresee its applications to domains such as, e.g., vector floor-plan generation [Shabani et al., 2023].

Limitations. During generation, we observed that the two neural networks composing GRIDDD may occasionally conflict, simultaneously performing an insertion and a deletion at the same timestep, which is theoretically illegal in our framework. GRIDDD also tends to produce more split molecules compared to FreeGress and DiGress. This is likely because nodes inserted late in the denoising process are challenging to connect before denoising concludes. Incorporating additional input features, such as the current number of connected components, could potentially mitigate this issue.

Acknowledgments

This work has been partially supported by EU-EIC EMERGE (Grant No. 101070918) and PNRR, PE00000013, "FAIR - Future Artificial Intelligence Research", Spoke 1, funded by European Commission under NextGeneration EU programme (CUP: B53D2302625000).

References

Andrew Campbell, William Harvey, Christian Weilbach, Valentin De Bortoli, Tom Rainforth, and Arnaud Doucet. Trans-dimensional generative modeling via jump diffusion models, 2023. URL https://arxiv.org/abs/2305.16261.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, volume 33, pages 13260–13271. Curran Associates, Inc., 2020.

Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=AAWuCvzaVt.

Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R. Rabiee. Deep graph generators: A survey. *IEEE Access*, 9:106675–106702, 2021. doi: 10.1109/ACCESS.2021.3098417.

Jiazhen He, Huifang You, Emil Sandström, Eva Nittinger, Esben Jannik Bjerrum, Christian Tyrchan, Werngard Czechtizky, and Ola Engkvist. Molecular optimization by capturing chemist's intuition using deep neural networks. *J. Cheminformatics*, 13(1):26, 2021. doi: 10.1186/S13321-021-00497-0. URL https://doi.org/10.1186/s13321-021-00497-0.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop DGMs Applications*, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

Emiel Hoogeboom, Víctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3D. In *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.

John J. Irwin and Brian K. Shoichet. Zinc - a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005. doi: 10.1021/ci049714+. URL https://doi.org/10.1021/ci049714. PMID: 15667143.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2323–2332. PMLR, 10–15 Jul 2018a.

Wengong Jin, Kevin Yang, Regina Barzilay, et al. Learning multimodal graph-to-graph translation for molecular optimization. In *ICLR*, 2018b.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Daniel D. Johnson, Jacob Austin, Rianne van den Berg, and Daniel Tarlow. Beyond in-place corruption: Insertion and deletion in denoising probabilistic models. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. URL https://openreview.net/forum?id=cAsVBUe1Rnj.

Mohamed Amine Ketata, Nicholas Gao, Johanna Sommer, Tom Wollschläger, and Stephan Günnemann. Lift your molecules: Molecular graph generation in latent euclidean space. In *The Thirteenth International Conference on Learning Representations*, 2025.

Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B. Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 17391–17408. PMLR, 2023. URL https://proceedings.mlr.press/v202/kong23b.html.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Constrained graph variational autoencoders for molecule design. In *Neural Information Processing Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:43924638.

Matteo Ninniri, Marco Podda, and Davide Bacciu. Classifier-free graph diffusion for molecular property targeting. In Albert Bifet, Jesse Davis, Tomas Krilavičius, Meelis Kull, Eirini Ntoutsi, and Indrė Žliobaitė, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 318–335, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-70359-1.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018. doi: 10.1609/aaai.v32i1.11671.

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), August 2014. ISSN 2052-4463. doi: 10.1038/sdata.2014.22.

Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5466–5475. IEEE, June 2023. doi: 10.1109/cvpr52729.2023.00529. URL http://dx.doi.org/10.1109/CVPR52729.2023.00529.

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*, 2023a.

Clement Vignac, Nagham Osman, Laura Toni, and Pascal Frossard. Midi: Mixed graph and 3d denoising diffusion for molecule generation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 560–576. Springer, 2023b.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a.

Jiaxuan You, Rex Ying, Xiang Ren, et al. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5708–5717. PMLR, 10–15 Jul 2018b.

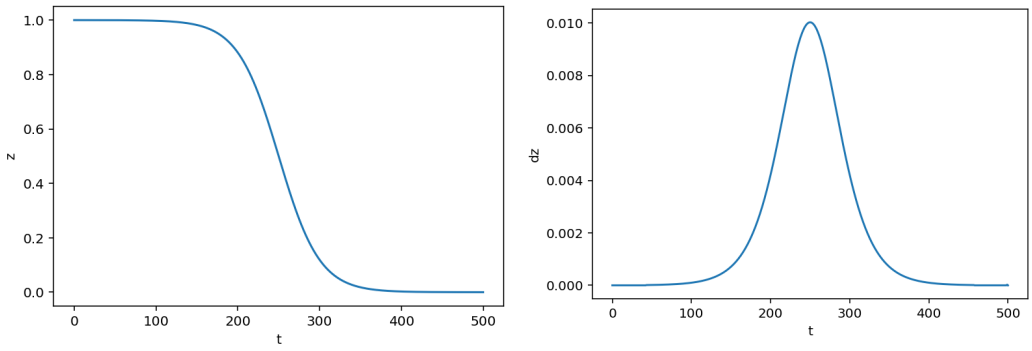
Yong Zhao, Edirisuriya M. Dilanga Siriwardane, Zhenyao Wu, Nihang Fu, Mohammed Al-Fahdi, Ming Hu, and Jianjun Hu. Physics guided deep learning for generative design of crystal materials with symmetry constraints. *npj Computational Materials*, 9(1), March 2023. ISSN 2057-3960. doi: 10.1038/s41524-023-00987-9. URL http://dx.doi.org/10.1038/s41524-023-00987-9.

Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. A survey on deep graph generation: Methods and applications. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=Im8G9R1boQi.

A Additional details on the methodology

A.1 Delete scheduler and insert/delete timestep distributions

We show a possible realization of the delete scheduler $\zeta(t)$ and the insert/delete timestep distribution $\zeta^{'}(t)$ in Figure 4.



(a) $\zeta(t)$ (b) $\zeta^{'}(t) = \frac{\delta\zeta(t)}{\delta t}$

Figure 4: The functions $\zeta(t)$ and its derivative $\zeta^{'}(t)$ for $w = 0.05$ and $T = 500$. $1 - \zeta(t)$ is effectively the "delete scheduler", and represents the probability that a node selected for deletion has already been deleted at timestep t. $\zeta^{'}(t)$ represents instead the probability that a node chosen for deletion will switch to category DEL* exactly at timestep t.

A.2 Distribution of the labels of the inserted nodes and edges

During the forward process, GRIDDD assigns labels to the newly inserted nodes and edges using a marginal distribution of these labels computed on the individual training sample. Such distributions can be computed just once before starting the training process and can also be saved for multiple training sessions. We have also tried different distributions during our preliminary studies, such as always inserting the least frequent node/edge class, as well as sampling them from a uniform probability, but the sample's marginal distribution gave us better results.

A.3 Training and sampling algorithms

Here, we provide the pseudocode describing the procedures to train (Algorithm 1) and sample from (Algorithm 2) GRIDDD.

B Additional experimental details

B.1 Dataset splits

QM9. On QM9, the training set is made of the first 100000 samples in the dataset. The test set is made of 10% of the overall data, and the remaining data is used to make the validation set.

ZINC-250k. The training set uses the first 80% of the data. The remainder is equally split between the validation set and the test set.

B.2 Neural architecture

Both the main model and the auxiliary model employed to predict the number of DELs are based on FreeGress' architecture. The model is composed of a stack of Graph Transformer layers. It receives as input the node features matrix $\boldsymbol{X}$, edge features matrix $\boldsymbol{E}$, and a vector $\boldsymbol{u}$ encoding the timestep t and optional extra features. These inputs are augmented with conditioning information $\boldsymbol{y}$ (denoted with the $+$ superscript in Figure 5a) and processed through the transformer stack.

https://doi.org/10.52202/085713-2625

Algorithm 1 Pseudocode of the training algorithm

Require: $\mathcal{G} = \{\boldsymbol{G}_i\}_{i=1}^{N}$, dataset of N graphs
Require: T, the number of denoising timesteps,
Require: p_θ, the main model,
Require: g_ϕ, the model tasked to predict the number of $\texttt{DEL}^*$s in $\boldsymbol{G}^t$,
Require: $\zeta^{'}(t)$, a discrete probability distribution function defined on T,
Require: $h(n)$, which given a graph size n returns a probability distribution h_n on the number of nodes.

for each graph $\boldsymbol{G}_i = (\boldsymbol{X}_i, \boldsymbol{E}_i) \in \mathcal{G}$: **do**
  $n^0 = |\boldsymbol{X}_i|$
  $n^T \sim h_{n^0}(n)$
  $t \sim Uniform(1, T)$
  $\boldsymbol{m}_{\boldsymbol{X}_i}, \boldsymbol{m}_{\boldsymbol{E}_i}$ = marginal distribution computed on the node and edges of $\boldsymbol{G}_i$.
  $\boldsymbol{S}^* = \{0, \dots, 0\}$ vector storing the insertion times of the nodes
  $\boldsymbol{X}_i^{*t}, \boldsymbol{E}_i^{*t} \sim (\boldsymbol{X}_i^t)^{'}\overline{Q}_{\boldsymbol{X}}^{t|0}, (\boldsymbol{E}_i^t)^{'}\overline{Q}_{\boldsymbol{E}}^{t|0}$
  $\Delta^T = n^T - n^0$
  Sample $\boldsymbol{U} = \{u_j\}$ timesteps, with $j \in \{0 \dots |\Delta^T| - 1\}$, where $u_j \sim \zeta^{'}$
  Remove from $\boldsymbol{U}$ the elements greater than t
  for each $u_j \in \boldsymbol{U}$ **do**
    if $\Delta^T > 0$ **then** ▷ Insert
      Sample a new node $\boldsymbol{x} \sim \boldsymbol{m}_{\boldsymbol{X}_i}$ with insertion time u_j, and add it to $\boldsymbol{X}_i$
      Sample $\boldsymbol{x}^t$ from the distribution $\boldsymbol{x}^{'}\overline{\boldsymbol{Q}}_{\boldsymbol{X}}^{*t|u_j}$ and add it to $\boldsymbol{X}_i^{*t}$
      Sample from $\boldsymbol{m}_{\boldsymbol{E}_i}$, for each node already in the graph, an inward and outward edge with the newly added $\boldsymbol{x}_t$
      Sample the edges $\boldsymbol{e}$'s final state $\boldsymbol{e}^t$ from $\boldsymbol{e}^{'}\overline{Q}_{\boldsymbol{E}}^{*t|u_j}$ and add them to $\boldsymbol{E}_i^{*t}$
      Append u_j at the end of $\boldsymbol{S}^*$
    else if $\Delta^T < 0$ **then** ▷ Delete
      Sample one random node $\boldsymbol{x}$ among the ones in $\boldsymbol{G}_i$ that are not set to state $\texttt{DEL}$ or $\texttt{DEL}^*$
      if $u_j = t$ **then**
        $\boldsymbol{x} = \texttt{DEL}^*$
        Set all incoming and outgoing edges of $\boldsymbol{x}$ as $\texttt{DEL}^*$
      else
        Remove $\boldsymbol{x}$ from $\boldsymbol{X}_i^{*t}$
        Remove all incoming and outgoing edges of $\boldsymbol{x}$ from $\boldsymbol{E}_i^{*t}$
        Remove the entry in $\boldsymbol{S}^*$ associated to the node
      end if
    end if
  end for
  $\boldsymbol{G}^{*t} = (\boldsymbol{X}^{*t}, \boldsymbol{E}^{*t})$
  $\{(\widehat{\boldsymbol{X}}^{*0}, \widehat{\boldsymbol{E}}^{*0}), \widehat{\boldsymbol{S}}^*\} = p_\theta(\boldsymbol{G}^{*t})$
  $loss_i = \lambda_X \mathrm{CE}(\widehat{\boldsymbol{X}}^{*0}, \boldsymbol{X}^{*0}) + \lambda_E \mathrm{CE}(\widehat{\boldsymbol{E}}^{*0}, \boldsymbol{E}^{*0}) + \lambda_S \mathrm{CE}(\widehat{\boldsymbol{S}}^{*0}, \boldsymbol{S}^{*0})$
  $n_{\texttt{DEL}^*}$ = number of nodes set to $\texttt{DEL}^*$ in $\boldsymbol{G}^{*t}$
  Remove all nodes set to $\texttt{DEL}^*$ in $\boldsymbol{G}^{*t}$ as well as the associated edges
  $loss_i^* = CE(g_\phi(\boldsymbol{G}^{*t}), n_{\texttt{DEL}^*})$
end for

Algorithm 2 Pseudocode of the denoising algorithm

Require: T, the number of denoising timesteps,
Require: n^T, the number of nodes in the noisy graph $\boldsymbol{G}^T$,
Require: p_θ, the main model,
Require: g_ϕ, the model tasked to predict the number of DEL*s in $\boldsymbol{G}^t$,
Require: $\boldsymbol{m_X}$ and $\boldsymbol{m_E}$, the dataset's marginal distributions on the nodes and the edges;
Sample $\boldsymbol{G}^T = (\boldsymbol{X}^T, \boldsymbol{E}^T)$ with n^T nodes from an the marginal distributions $\boldsymbol{m_X}$ and $\boldsymbol{m_E}$;
for each timestep $t \in [T, \dots, 1]$: **do**
    Use $g_\phi(\boldsymbol{G}^t)$ to predict the number of DEL* entries to insert in the graph, and insert them.
    Set all the edges connecting the newly inserted nodes to the rest of the graph as DEL*.
    Use $p_\theta(\boldsymbol{G}^t)$ to predict $\boldsymbol{S}^t$, the insertion times for each node in the graph, and $\boldsymbol{G}^{*0}$.
    Remove the nodes $\boldsymbol{x}_i$ such that $\boldsymbol{S}_i = t$, as well as their edges.
    Use $\boldsymbol{S}^t$ and $\boldsymbol{G}^{*0}$ to sample, for each node $\boldsymbol{x}_i^t$ and edge $\boldsymbol{e}_{ij}^t$ in $\boldsymbol{G}^t$, their type at step $t-1$:
    $\boldsymbol{x}_i^{t-1} \sim p_\theta(\boldsymbol{x}_i^{t-1}|\boldsymbol{x}_i^t) = \sum_{x \in \chi} \frac{q(\boldsymbol{x}_i^t|\boldsymbol{x}_i^{t-1}) q(\boldsymbol{x}_i^{t-1}|\boldsymbol{x}_i^{S_i}=x)}{q(\boldsymbol{x}_i^t|\boldsymbol{x}_i^{S_i}=x)} p_\theta(\boldsymbol{x}_i^{S_i} = x|\boldsymbol{x}^t)$.
    $\boldsymbol{e}_{ij}^{t-1} \sim p_\theta(\boldsymbol{e}_{ij}^{t-1}|\boldsymbol{e}_{ij}^t) = \sum_{\boldsymbol{e} \in \varepsilon} \frac{q(\boldsymbol{e}_{ij}^t|\boldsymbol{e}_{ij}^{t-1}) q(\boldsymbol{e}_{ij}^{t-1}|\boldsymbol{e}_{ij}^{max(S_i,S_j)}=\boldsymbol{e})}{q(\boldsymbol{e}_{ij}^t|\boldsymbol{e}_{ij}^{max(\boldsymbol{S}_i,\boldsymbol{S}_j)}=\boldsymbol{e})} p_\theta(\boldsymbol{e}_{ij}^{max(S_i,S_j)} = \boldsymbol{e}|\boldsymbol{e}_{ij}^t)$.
    $\boldsymbol{G}^{t-1} = (\{\boldsymbol{x}_i^{t-1}\}, \{\boldsymbol{e}_{ij}^{t-1}\})$
end for

Each Graph Transformer layer (Figure 5b) follows the standard architecture: self-attention, dropout, residual connection, layer normalization, and a feedforward block. The self-attention mechanism (Figure 5c) applies scaled dot-product attention to the augmented node features $\boldsymbol{X}$. The resulting attention weights are modulated by the edge tensor $\boldsymbol{E}$ via a FiLM layer [Perez et al., 2018], normalized with softmax, and used to re-weight $\boldsymbol{X}$. The output is flattened and combined with $\boldsymbol{u}$ through another FiLM layer before linear projection to prediction logits.

Separately, $\boldsymbol{X}$ and $\boldsymbol{E}$ are passed through independent PNA layers [Corso et al., 2020], and their outputs are summed with a linearly projected $\boldsymbol{u}$ to form the final graph representation. In GRIDDD, the final output $\boldsymbol{u}'$ is employed by our auxiliary model to predict the amount of DEL* nodes that must be reintroduced at the current timestep. The final Graph Transformer Layer's output designed to predict the matrix $\boldsymbol{X}'$ is passed to two Linear layers, instead of just one. The first is used, just like before, to predict the final node matrix. The second one is used instead to predict the activation timestep s of the various nodes.

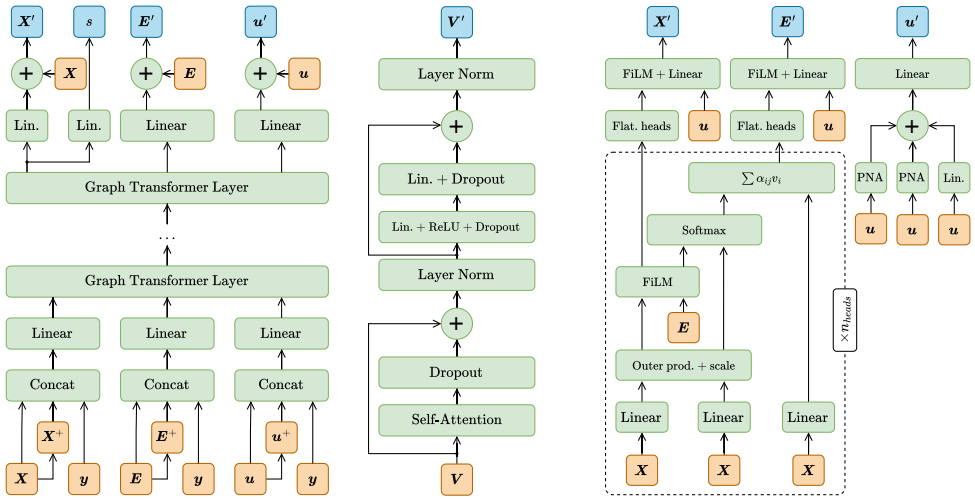
B.3 Baselines

For the task of property targeting, discussed in Section 4.1, we compared our model against DiGress [Vignac et al., 2023a], a Graph Diffusion Model capable of performing conditional graph generation through a classifier-based guidance, and FreeGress [Ninniri et al., 2024], a Graph Diffusion Model which employs a classifier-free guidance system instead. For the task of property optimization, discussed in Section 4.2, we employ a baseline of three models. In particular, we employ the Junction-Tree Variational Autoencoder (JT-VAE, Jin et al. [2018a]), an Autoencoder that works on the molecular sub-structural level rather than the atomic level, the Graph Convolutional Policy Network (GCPN, You et al. [2018a]), which employs Reinforcement Learning to generate molecules atom-by-atom, and the Constrained Graph Variational Autoencoders (CG-VAE, Liu et al. [2018]), a graph-based Autoencoder whose peculiarity is that it applies chemical validity constraints during the generative process to increase its performance in molecular tasks.

B.4 Considerations on the independence between nodes and edges in the forward process

When a node is switched to DEL*, so are the edges associated with it. This effectively creates a dependence between the two, while standard Diffusion Models assume a mutual independence between them. This is similar to what is done with Masked Diffusion [Kong et al., 2023], where the edges associated with masked nodes are masked as well. In our case, what we effectively do is replacing the forward process $p(\boldsymbol{e}_{ij}^t|\boldsymbol{e}_{ij}^{t-1})$ with $p(\boldsymbol{e}_{ij}^t|\boldsymbol{e}_{ij}^{t-1}, \boldsymbol{x}_i^t, \boldsymbol{x}_j^t)$, and $p(\boldsymbol{e}_{ij}^{t-1}|\boldsymbol{e}_{ij}^t, \boldsymbol{e}_{ij}^0)$ with $p(\boldsymbol{e}_{ij}^{t-1}|\boldsymbol{e}_{ij}^t, \boldsymbol{e}_{ij}^0, \boldsymbol{x}_i^t, \boldsymbol{x}_j^t)$. However, it should be kept in mind how the latter equation is, in all practical

https://doi.org/10.52202/085713-2625



(a) (b) (c)

Figure 5: (a) High-level architecture of GRIDDD. (b) Detail of the graph transformer layer. $\boldsymbol{V}$ represents the triple $(\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{u})$. (c) The self attention-layer within the graph transformer layer. In the pictures, orange boxes indicate inputs, blue boxes indicate outputs, and green boxes indicate layers/operations.

scenarios, identical to $p(\boldsymbol{e}_{ij}^{t-1}|\boldsymbol{e}_{ij}^{t}, \boldsymbol{e}_{ij}^{0})$, as the only situations where $\boldsymbol{x}_{i/j}^{t}$ could influence the value of $\boldsymbol{e}_{ij}^{t-1}$ is when the former is labeled as DEL* and the latter to DEL, in which case $\boldsymbol{e}_{ij}^{t-1}$ should switch to DEL* as well. However, this scenario is not possible in practice, as neither the nodes nor the edges can appear explicitly with label DEL during the reverse process. We leave to future research the study of independent edge processes that still allow for the deletion of the associated nodes.

C Additional computational considerations

C.1 Runtime

On average, one training epoch of GRIDDD takes 309.1 seconds on two nVidia A100 GPUs. FreeGress, on the same problem (hence, without insert and delete operations), takes 243 seconds. This means that GRIDDD is approximately 27% slower than its counterpart. This is similar to the 33.3% increase we have found on QM9, where GRIDDD takes 18 seconds on average to complete a training epoch on a single A100, while FreeGress takes 13.5.

C.2 Sampling speed

Interestingly enough, GRIDDD can be faster at sampling than standard Diffusion Models. When tasked with generating a variegated batch of graphs with different sizes, a standard Diffusion Model would first sample the graph sizes from the dataset's node distribution. This means that, in practice, the sizes of the node and edge matrices will have one or more dimensions equal to the highest graph size sampled. In turn, this means that every graph generated will require a processing power proportional to such size.

On the other hand, GRIDDD is capable of generating a graph of any shape, even when starting from a different size. In this scenario, one could start the sampling process with very small latents (even zero nodes). Since the tensors representing the sample batch would be very small, the computational power required to process these tensors will be small as well. This means that the early phases of the sampling process, before most nodes are inserted, will likely be faster than standard diffusion (although this is highly dependent on $\zeta(t)$).

To test this hypothesis, we have sampled 128 molecules on ZINC-250K, conditioning on the LogP value, with FreeGress and GRIDDD. For the latter, we have performed three experiments. In the first,

GRIDDD starts from latents with a size of two nodes. In the second, we have used an initial size of 24 (the most frequent graph size in the training set), and then we have sampled the initial graph size using the training set's node distribution, as is done in FreeGress. The results, summarized in Table 5, show how GRIDDD is capable of performing as well as FreeGress while halving the sampling time in the best case scenario, and even in the worst case scenario it requires as little as five percent more time to perform as well as FreeGress.

Table 5: Sampling speed comparison between GrIDDD and FreeGress when generating 128 samples on ZINC-250k conditioned on the LogP value, using different initial latent sizes.

Model	**Validity** ↑	**MAE** ↓	**Sampling time** ↓
FreeGress	89.8%	0.31	152.05s
GRIDDD (initial size=2)	**91.4%**	**0.23**	**68.09*s***
GRIDDD (initial size=24)	90.6%	0.31	120.19s
GRIDDD (initial size $\sim p(n)$)	88.2%	0.26	159.85s

D Additional results

D.1 Comparison with EDM-SyCO

We compared to the 3D Diffusion Model by Ketata et al. [2025] called EDM-SyCO, since it performs property optimization similarly to GRIDDD. Notably, EDM-SyCO also inputs 3D coordinates to the diffusion process. To ensure a fair comparison, we slightly adapted our original setup to the less restrictive one used in the related paper. In particular, they attempt to optimize each molecule 100 times, rather than 20 as in our previous experiment. Then, among the ones within the similarity threshold, they select the 10 ones with the best improvement, duplicate each one of them ten times, and run the optimization process a second time. The process is repeated four times (for a total of 400 optimization rounds), after which the molecule with the best improvement within the similarity threshold is selected. All the successful molecules obtained in the process (that is, within the similarity threshold) are considered when computing the diversity score. In cases where the number of successful experiments is less than two, the diversity score is set to zero. Results of the comparison are reported in Table 6, showing competitive or superior performances by GRIDDD, even though it does not benefit from using 3D information as input. We have compared our results against EDM-SyCo on property targeting as well. Specifically, we have trained a model on ZINC-250K with 1000 denoising timesteps (as is done by our baseline) that is conditioned on the molecular weight. We have obtained a MAE of 2.13 ± 0.19 and a validity of 86.2%, while EDM-SyCo records a MAE of 3.86 ± 0.08 and a validity of 88%.

Table 6: Comparison with EDM-SyCo on property optimization.

Method	**LogP (sim $\geq$ 0.4)**		**LogP (sim $\geq$ 0.6)**		**QED (sim $\geq$ 0.4)**		**DRD2 (sim $\geq$ 0.4)**	
	Improv. ↑	**Div.**↑	**Improv.** ↑	**Div.**↑	**Succ.**↑	**Div.**↑	**Succ.**↑	**Div.**↑
EDM-SYCO	3.11 ± 1.27	**0.555**	1.51 ± 1.10	**0.360**	46.4 %	0.163	**27.3** %	**0.083**
GRIDDD	**3.27 ± 0.91**	0.511	**1.59 ± 0.54**	**0.359**	**64.1** %	**0.269**	19.7 %	0.058

D.2 Hyperparameter sensitivity analysis

We show in Table 7 an analysis of GRIDDD's sensitivity of the hyperparameters regulating the scheduler (D and w) and the node counts distribution (p_{min} and p_{max}). One interesting insight that can be gathered from the table is the fact that the number of split molecules generated significantly increases with smaller Ds. From a practical perspective, small values for this hyperparameter imply that, during the denoising process, the model can insert nodes relatively late during the reverse process. We conjecture that split molecules are mostly caused by nodes inserted too late during such a process, as they cannot be re-attached in time to the main graph since the noise scheduler does not allow for too many changes in the graph's structure during the last phases of the process. We

 https://doi.org/10.52202/085713-2625

have also noted that this phenomenon is sensibly reduced when GRIDDD is given in input, as extra features, different powers of the adjacency matrix.

Table 7: Study of GrIDDD's sensitivity to hyperparameter changes when sampling 100 molecules after training on the QM9 dataset. The hyperparameters tested are the delete scheduler's parameters D, w, and the node count distribution's parameters p_{min} and p_{max}. Val: validity, Avg NC: average number of connected components, Max NC: maximum number of connected components, NSC: number of graphs sampled with a single connected component.

D	**Val** ↑	**Avg NC** ↓	**Max NC** ↓	**NSC** ↑
0.25	0.93	1.15	4	88
0.50	0.93	1.02	2	98
0.75	0.95	1.01	2	99

w	**Val** ↑	**Avg NC** ↓	**Max NC** ↓	**NSC** ↑
0.025	0.94	1	1	100
0.050	0.93	1.02	2	98
0.075	0.92	1.03	2	97

p_{min}	**Val** ↑	**Avg NC** ↓	**Max NC** ↓	**NSC** ↑
0.2	0.93	1.02	2	98
0.4	0.91	1	1	100
0.6	0.96	1.02	3	99
0.8	0.94	1.02	2	98

p_{max}	**Val** ↑	**Avg NC** ↓	**Max NC** ↓	**NSC** ↑
0.5	0.86	1.02	2	98
0.1	0.93	1.02	2	98
0.01	0.89	1.01	2	99
0.001	0.94	1.02	2	98

D.3 Generated molecules

Figure 6 shows 6 (3 for the QED task, 3 for the LogP task) randomly selected molecules optimized by GRIDDD on the ZINC-250k dataset. For each molecule, we show 4 different successful optimizations. Figure 7 shows 30 random molecules generated without conditioning by GRIDDD on QM9.

D.4 Failure Analysis

While analyzing the situations where GRIDDD fails in conditional generation the most, we have noticed how, unlike FreeGress, the experiments targeting μ tend to fail most frequently when targeting values close to zero. The few molecules generated tend to be small, split graphs. We have investigated the phenomenon and found out that the molecule with SMILES string `CC` (ethane) is known to have a dipole moment equal to zero. We believe that it is likely that GRIDDD tries to generate ethane or similar small molecules to minimize μ, with higher failure rates than usual since these small molecules are under-represented in the dataset. Remarkably, DiGress and FreeGress are unable to pursue this minimization, since they almost inevitably start the reverse process from latents with a larger graph size and cannot adapt it to produce smaller molecules later on. Overall, this is an interesting behavior which shows that GRIDDD can successfully learn the properties of molecules with infrequent sizes as well.

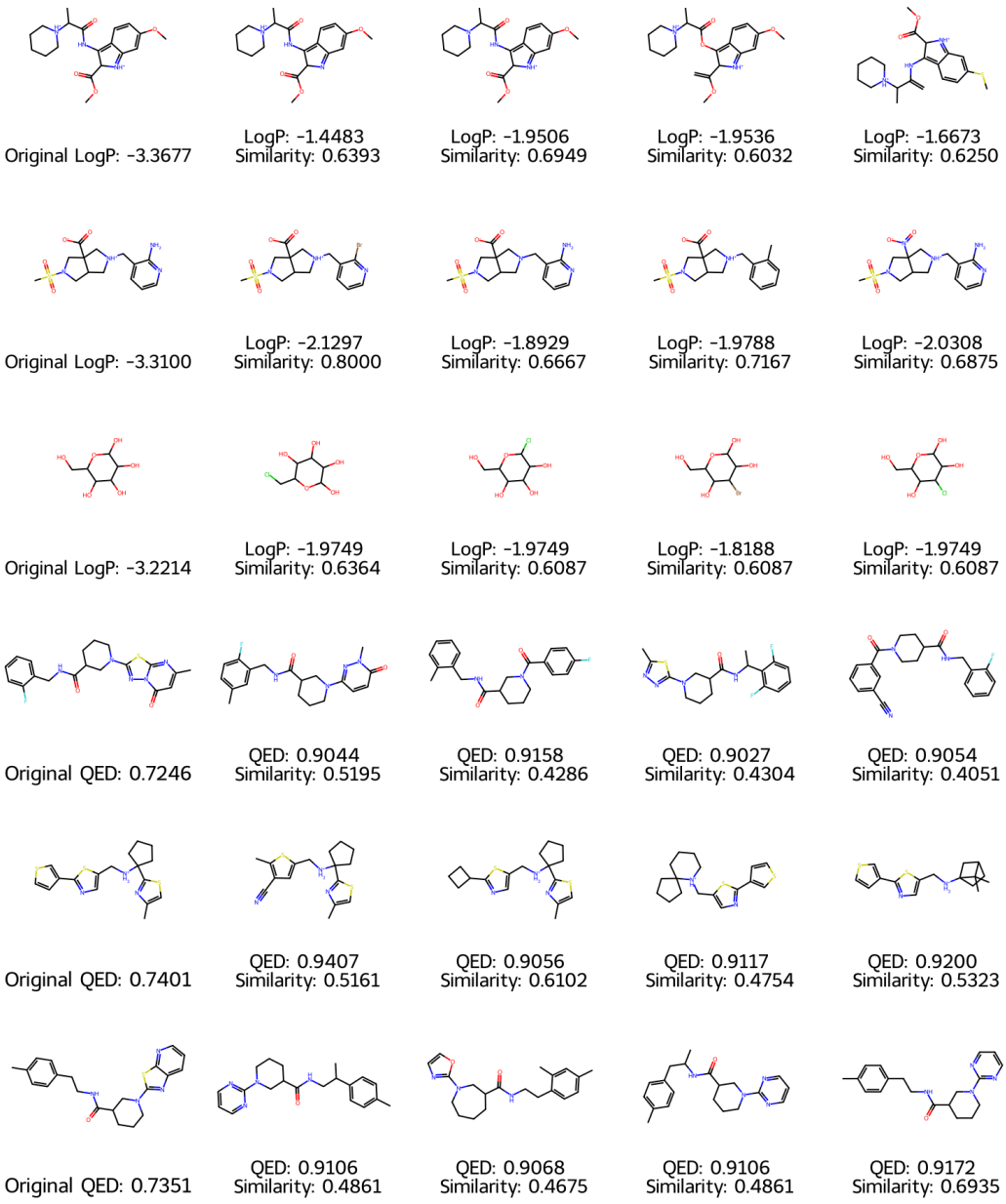


Figure 6: Randomly selected optimization experiments on ZINC-250k. The first element in each row is the original molecule, while the other four are results of different optimization experiments. We report, for each experiment, the original value and the resulting one.

https://doi.org/10.52202/085713-2625

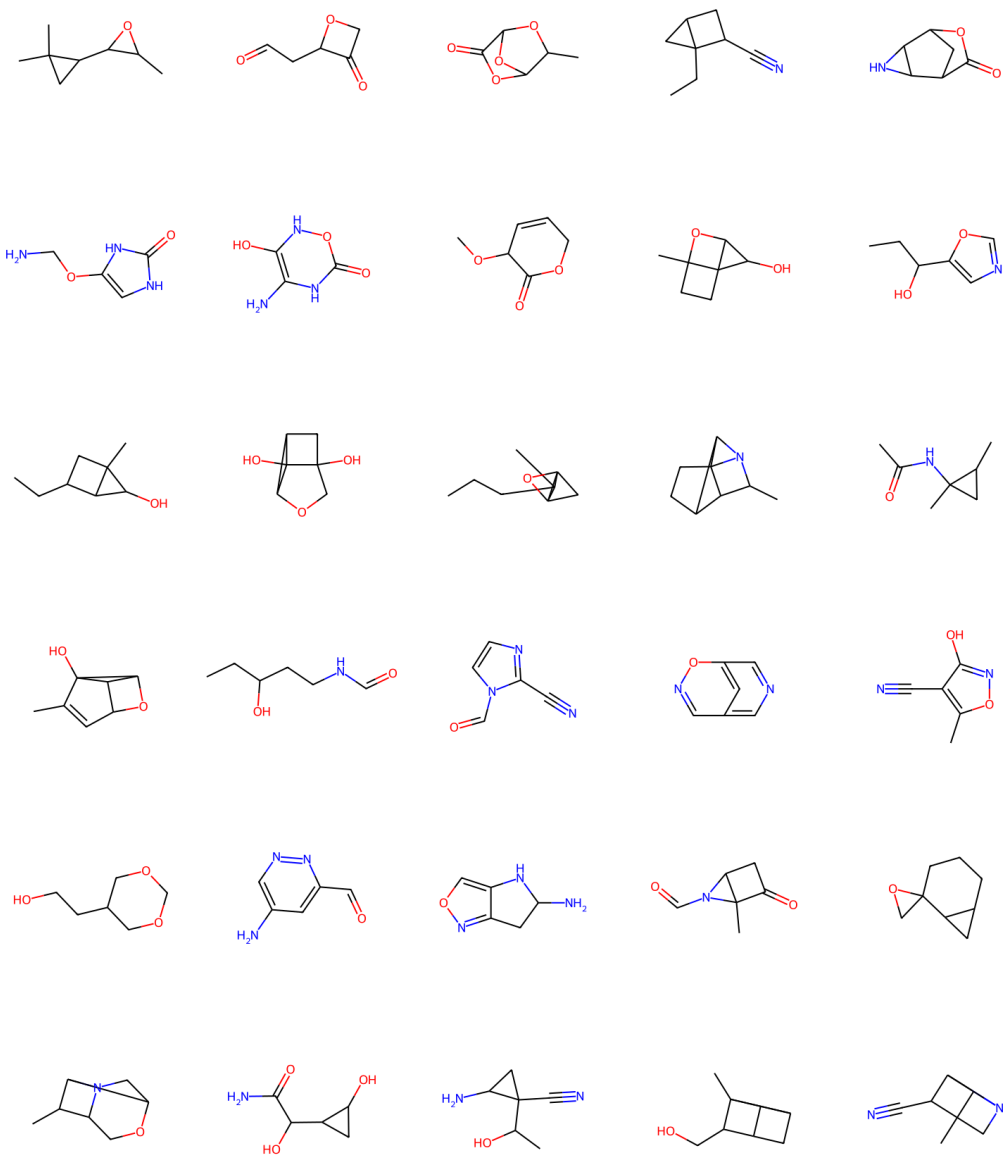


Figure 7: 30 non-curated samples generated without conditioning on QM9.

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 have introduced a new model called GRIDDD, which can perform insert and delete operations. The Results section shows how it can match and even surpass the baseline used despite being trained on an arguably more difficult problem.

   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 have discussed the limitations of our model in the Conclusions. We have been as transparent as possible.

   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**

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

https://doi.org/10.52202/085713-2625

Answer: [NA]

Justification: Our paper does not include theoretical results.

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 have been as complete as possible. Our experiments employ the same architecture of previously published models with the same hyperparameters (albeit with minor modifications discussed in the paper). Regardless, we have discussed the architecture in detail in the supplemental material.

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 include the source code, as well as instructions on how to set up a Conda environment to run it, both in the supplemental material and in the GitHub repository referenced in the main document.

   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 have discussed the dataset splits in the supplemental material. As stated in the main paper, most of our hyperparameters are the same used by FreeGress, and we have only listed the differences with our solution. Our source code contains the various configurations files in .yaml format.

   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 reported the same metrics used by the baseline.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

https://doi.org/10.52202/085713-2625

- 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 discussed the type of GPU employed. In the appendix, we discussed the run time compared to the baseline. The runtime is similar for all experiments, so we did not find it necessary to report every single run time.

   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: We believe our work is conforming to the Code of Ethics.

   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: We do not believe that our work will have any societal impact.

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: Our work 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: We have credited the paper on which we based our own code. We have cited both ZINC-250k and QM9. We are not aware of any license or terms of use concerning these assets.

    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.
    - The authors should state which version of the asset is used and, if possible, include a URL.

 https://doi.org/10.52202/085713-2625

- 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: [NA]

    Justification: We did not release any new asset.

    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: Our work does not involve crowdsourcing and we did not perform 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: Our work does not involve crowdsourcing and we did not perform 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.

16. **Declaration of LLM usage**

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

    Answer: [NA]

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

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

https://doi.org/10.52202/085713-2625